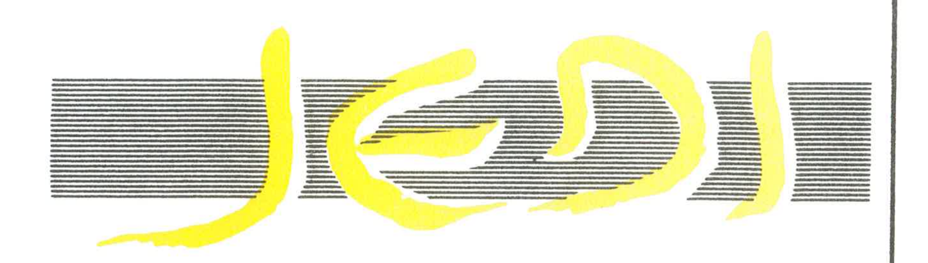


58

ENCORE 152 NUMEROS AVANT L'AN 2000

AVRIL 1987

EDITORIAL

La collaboration entre JEDI et FIG HAMBOURG devient effective avec la diffusion par JEDI de la version VolksFORTH83. Ainsi, avant même que l'idée ne soit lancée par les politiciens européens avons nous unis les FORTHs français aux FORTHs allemands (arf, arf!! Plus on est de fous...). C'est pouquoi les illustrations de JEDI sont inspirés de la rock-culture germanique (KRAFTWERK en couverture si vous n'avez pas reconnu). SOuhaitons que l'on puisse appliquer à VolksFORTH83 le même slogan qu'à la coccinelle de chez Volkswagen: "née pour durer".

Et puisque nous en sommes à parler de nos voisins d'OutreRhin, saluons l'exploit de ce jeune pilote qui a atteri sur la place rouge. Le risque était grand, mais à travers cet acte on peut se poser quelques questions concernant la défense d'un territoire, que ce soit l'URSS ou tout autre pays. Une initiative de défense stratégique (guerre des étoiles) n'estelle pas caduque si n'importe quel engin autre qu'un missile peut pénétrer sur un territoire étranger. Le cas du jeune allemand n'est pas unique. Nous avons eu pour notre part, en France, une histoire d'évasion par hélicoptère de la prison de la Santé alors que le survol de la capitale est interdit!!.

Le thème de la bombe atomique 'sous le paillasson' a déjà été exploité dans de nombreux romans (un baril de carburant dans le CINQUIEME CAVALIER, un avion de lignes régulières dans '30 SECONDES SUR NEWYORK,...). A quand le camping car nucléaire?

Soyons humbles, Mathias RUST a rendu plutôt service en rendant évident les défaut de la cuirasse: la routine et la complexité de la tâche de surveillance. En conclusion, si les hommes sont faillibles, que peutil en être des systèmes experts qui seront chargés d'assurer la défense dans le projet IDS américain?

SOMMAIRE

FORTH: QUESTIONS/REPONSES: R4 R8 — 6
les questions ne se bousculent guère. Vous avez donc tout compris de FORTH? (et des autres langages.

UN METACOMPILATEUR SIMPLE — 7
pour installer FORTH83 sur système 6502, le principe de base.

ASSEMBLEUR 6809 EN FORTH — 10
au cinéma on dit 'LE FILM', ici on parle du 'LISTING'. Bonne projection!

FBASEII pour ORIC ATMOS — 15
une adaptation F83 de fBASE 1 avec des améliorations mâtin!!

VolksFORTH83 pour ATARI 260, 520ST, 520ST+ — 17
un Forth made in Germany public domain.

dBASE II PRESENTATION — 2
il était temps qu'on en parle.

I.A. REDACTOR — 4
créez votre société en évitant les embûches juridiques...

Ont participé à l'élaboration de ce numéro, dans le désordre: Marc PETREMANN, Edouad SAINTE-MARIE, Yann DESBOIS, A.JACCOMARD, Guy T.GROTKE, Guy M.KELLY, Jean-Luc SIRET, FIG HAMBURG.

Nos coordonnées: ASSOCIATION JEDI 17, rue de la Lancette 75012 PARIS
tel président: (1) 43.40.96.53
tel secrétaire:(1) 46.56.33.67

dBASE II
NI LANGAGE, NI PROGRAMME
LA NOTION DE PROGICIEL
ET LA GESTION DE BASES DE DONNEES

par Marc PETREMANN

La gestion des données reste le point faible de la majorité des langages informatiques. De cette lacune est née une nouvelle génération de produits, ni logiciels, ni langages, les PROGICIELS spécialisés dans la gestion de bases de données dont dBASE est le meilleur exemple d'illustration de ce nouveau concept.

PREMIERE APPROCHE D'UN PROGICIEL

Le fonctionnement de tout système informatique nécessite la présence d'un certain nombre d'accessoires ou périphériques, et de logiciels de base. La palette minimale du gestionnaire est constituée par un tableur, un traitement de texte et un système de gestion de base de données. Pour être efficace, le gestionnaire doit connaître quelques rudiments de programmation, mais pas question pour lui de perdre son temps à programmer. Ceux qui ont essayé de gérer un fichier en Basic comprendront, non que Basic soit un mauvais langage, mais les outils nécessaires sont longs à programmer et difficiles à mettre en oeuvre.

De même, il n'est pas question d'utiliser une application toute faite, sauf si vous destinez votre micro à une tâche très spécifique. Il faut donc envisager l'acquisition d'un outil alliant simplicité et efficacité.

L'outil logiciel répondant le mieux à nos désirs se nomme dBASE. Sous ce nom se cache une famille de logiciels qui a fait ses preuves sur une large gamme de matériel. D'abord implanté sur la gamme CP/M, la version dBASE II est devenue dBASE III et dBASE III+ sous MSDOS. Dans la suite de notre propos, nous n'allons pas entrer dans les détails des différentes versions, mais nous attacher plus particulièrement à illuster en priorité le concept dBASE commun à cette gamme de produits.

Le système de gestion de données dBASE dispose d'un interpréteur donnant accès à des fonctions puissantes:

- création et définition d'un fichier
- sélection multicritère
- tri et/ou indexation
- impression sur écran/imprimante/disque

Il dispose également d'un éditeur plein écran simplifié permettant la création d'un fichier de commandes. Mais pour de grosses applications, on utilisera de préférence un traitement de texte.

Un utilitaire nommé ZIP permet de créer un fichier de commande ou de format à partir d'un écran de saisie.

La version MSDOS peut être équipée en option d'un compilateur, d'un module mathématique, d'un traducteur transformant un fichier de commandes dBASE en langage C.

Les données gérees par dBASE proviennent le plus généralement d'un fichier créé par dBASE, mais il peut aussi gérer un fichier créé sous BASIC, PASCAL, FORTH et même d'un fichier créé manuellement à partir d'un programme de traitement de texte comme WORD ou WORDSTAR.

Le programme dBASE gère les fichiers de données (.DBF), d'index (.NDX), de commandes (.CMD) et de format (.FMT). Il permet également l'édtion de rapports simples à partir de la commande REPORT.

LA CREATION D'UN FICHIER DE DONNEES

Une fois dBASE activé, la première opération est la création d'un fichier. Pour ce faire, il suffit de taper CREATE <nom> où <nom> est le nom du fichier à créer. Prenons pour exemple, car il n'y a rien tel que les exemples, le cas d'un directeur d'école souhaitant réaliser quelques statistiques sur la notation des élèves de son établissement (ici petit coup de coude aux directeurs d'écoles essayant de réaliser ce petit travail sur du matériel ne disposant pas de dBASE, sadisme oblige...). Pour ce faire, il lui faut le nom des élèves, la classe de l'élève et la note moyenne dans une matière générale. Il définira donc son fichier nommé NOTES comme suit:

```
STRUCTURE FOR FILE:     NOTES
NUMBER OF RECORDS:      00000
DATE OF LAST UPDATE:    00/00/00
PRIMARY USE DATABASE
FLD    TYPE WIDTH  DEC     NAME
001    C    20             ELEVE
002    C    3              CLASSE
003    N    2      0       NOTE
```

On supposera une notation sur 20. Nous réserverons 20 caractères pour le nom, 3 caractères pour la classe (CPE, CE1, CE2, etc...). Dès que la structure est définie, dBASE propose de commencer la saisie. Donc, si vous savez ce que vous désirez comme fichier, même avec 20 champs (ou plus), vous pouvez commencer une saisie dans la minute qui suit la création du fichier.

La saisie des données est présentée de manière simplifiée, mais autorise les manipulations 'plein écran'. L'option d'insertion est activée/désactivée par appui sur CTRL-V. Exemple d'écran de saisie:

```
#       1
ELEVE       :ADALBERT            :
CLASSE      :CPE:
NOTE        :12:
```

Le numéro figurant sur la première ligne correspond à la clé de l'enregistrement en cours dans le fichier. Avec un peu de patience, notre directeur cobaye rentre ses quatre cents références (ou moins pour une école qui n'est par surchargée, ...ou plus...). Corsons un peu le problème en signalant qu'il pointe les dossiers dans un ordre quelconque.

TRIER OU INDEXER

En premier lieu, l'opérateur peut vouloir remettre certaines références dans l'ordre alphabétique. Pour ce faire, il a le choix entre le tri et l'indexation. Le tri est lancé en tapant la commande SORT:

```
SORT ON ELEVE TO NOTES2
```

ce qui va créer un fichier NOTES2 trié par référence ELEVE. L'utilisation de ce fichier est activée par:

```
USE NOTES2
```

Si votre fichier est d'une taille conséquente, l'opération de tri risque de prendre quelque temps. De plus, le fichier généré ne peut être le fichier courant et sa taille sera identique au fichier initial. Si notre directeur souhaite un fichier trié par nom d'élève et un autre trié par notes, il devra générér deux autres fichiers.

L'indexation permet de générer un index se rapportant au seul champ servant de critère pour le tri. L'indexation par nom est réalisée par:

```
INDEX ON ELEVE TO NOMS
```

ce qui génère un fichier NOMS.NDX de taille bien inférieure au fichier de référence. Un autre fichier d'index par note est créé en tapant:

```
INDEX ON NOTE TO NOTES
```

EDITION PAR CRITERE

Dès maintenant, et toujours sans taper une seule ligne de programme, on peut éditer le contenu de notre fichier en totalité ou partiellement:

```
USE NOTES
LIST
```

liste la totalité du contenu du fichier NOTES

```
LIST ELEVE,CLASSE FOR NOTE>10
```

→

affiche la liste des élèves et la classe des éléments ayant une note supérieure à 10.

Cette même liste peut être obtenue par ordre alphabétique en activant l'index:

```
USE NOTES INDEX NOMS
LIST ELEVE FOR CLASSE='CPE' .AND. NOTE>10
```

affiche la liste des élèves par ordre alphabétique travaillant en classe CPE et ayant une note supérieure à 10.

Pour plus de clarté, la condition peut être parenthèsée:

```
LIST ELEVE FOR CLASSE='CM1' .AND. (NOTE>10 .AND. NOTE<=15)
```

affiche dans les mêmes conditions que précédemment le nom des élèves ayant une note supérieure à 10 et inférieure ou égale à 15.

FAIRE UN RAPPORT

A partir de ces éléments, notre cher directeur qui a l'esprit perfectionniste, exprime le voeu d'établir des rapports plus présentables afin de les soumettre à son académie pour soutenir une demande de budget. La commande REPORT sera utilisée de la manière suivante:

```
REPORT FORM TABLEAU
ENTER OPTIONS,M=LEFT,L=LINES/PAGE,W=PAGEWIDTH
PAGE HEADING? (Y/N) N
DOUBLE SPACE REPORT? (Y/N) N
ARE TOTAL REQUIRED? (Y/N) N
COL     WIDTH,CONTENTS
001     22,ELEVE
ENTER HEADING: NOM;===
002     8,CLASSE
ENTER HEADING: CLASSE;=====
```

ce qui peut être suffisant pour un premier essai. L'édition du contenu d'un rapport peut être paramétrée:

```
USE NOTES INDEX NOMS
REPORT FORM TABLEAU FOR NOTE<10 .AND. CLASSE='CPE'
```

ce qui n'est guère plus compliqué que du Basic, mais dont le résultat aurait demandé infiniment plus de temps à programmer dans ce même langage.

CREER UN FICHIER DE COMMANDES

Mais les exigences de notre vénéré directeur s'affinent avec ses compétences de gestionnaire de base de données. Il souhaite maintenant exploiter ces données à partir d'un programme de traitement de texte. Pour ce faire, il doit d'abord créer un fichier de commande réalisant une édition des différents rapports, donc écrire ce programme en dBASE. Ici apparaît une notion nouvelle, celle de langage informatique. Tout comme un langage informatique classique (BASIC, PASCAL, FORTH, C...) un progiciel dispose d'outils exécutables à partir de commandes, mais ces commandes peuvent être utilisées au sein d'un fichier regroupant une suite de commandes:

```
MODIFY COMMAND EDITION
```

crée un fichier de commande EDITION et passe en mode éditeur:

```
SET TALK OFF
SET SCREEN ON
SET ALTERNATE TO RAPPORTS.TXT
* ceci est le nom du fichier texte généré
SET ALTERNATE ON
USE NOTES INDEX NOMS
* premier rapport
? "NOTES DES ELEVES DE CPE"
?
REPORT FORM TABLEAU FOR CLASSE='CPE'
?
? "NOTES DES ELEVES DE CEL1"
?
REPORT FORM TABLEAU FOR CLASSE='CE1'

etc...
SET ALTERNATE OFF
```

et le contenu de ce fichier de commande est exécuté en tapant sous dBASE la commande:

```
DO EDITION
```

ce qui vous laisse le temps de vous faire un petit café (un ordinateur connait aussi le temps partagé). Une fois l'édition terminée, vous disposez d'un fichier RAPPORTS.TXT pouvant être retouché et complété à l'aide d'un traitement de texte classique.

Le progiciel dBASE dispose de structures simplifiées et utilisables à la manière de celles existantes en PASCAL. Toutes les commandes exécutables en mode interprété peuvent être utilisées dans un fichier de commandes.

La commande DO <fichier.CMD> permet l'exécution du contenu d'un autre fichier de commandes à partir du fichier de commande courant. La taille d'un fichier de commande est quelconque, dBASE exécutant séquentiellement le contenu des différents fichiers appelés.

La commande QUIT TO <fichier.COM> permet de lancer un programme autre qu'un fichier de commande, puis de revenir à la fin de son exécution pour reprendre l'interprétation du fichier de commandes. Un même fichier de commande dBASE peut ainsi exécuter un programme compilé en PASCAL, FORTH, C ou assembleur.

En théorie, le langage dBASE permet de définir n'importe quel type d'application spécialisée dans le traitement de données:

- mailing
- comptabilité
- gestion de stocks
- facturation

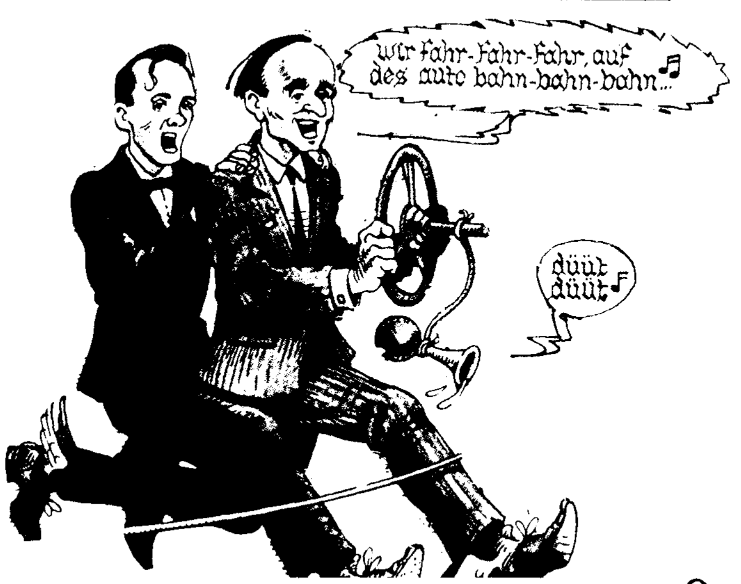
etc...

Nous avons même eu connaissance d'une application dBASE gérant un jeu d'aventure!

CONCLUSION

Destiné d'abord au domaine professionnel, le progiciel dBASE est devenu accessible notamment grâce à l'appartition de matériel performant à prix abordable. Si l'achat de dBASE reste encore un investissement, surtout en ce qui concerne dBASE III+, il est largement compensé par l'énorme facilité d'utilisation. D'un prix plus abordable, la version dBASE II, disponible sous CP/M et MSDOS, bien que moins performante, sera l'outil idéal pour réconcilier l'utilisateur ayant de petits fichiers à gérer ou des projets plus ambitieux.

De part sa très large diffusion, dBASE peut être considéré comme un langage à part entière. Diverses revues lui consacrent régulièrement des rubriques de programmes et tours de main. ■



REDACTOR, Le concept d'Intelligence Artificielle appliqué au domaine juridique

par
Edouard SAINTE-MARIE
et Yann DESBOIS

Nous avons exposé à AVOCAT 86 un programme, nommé REDACTOR, que Michel ROUSSEAU nous a proposé de présenter dans les colonnes de JEDI.

L'intérêt de REDACTOR réside en ceci qu'il est un système de communication utilisant comme unique règle de procédure celle de la grammaire d'un langage humain.

Nous avons étudié REDACTOR et fondé toute notre stratégie sur une analyse particulière du concept d'intelligence artificielle que nous considérons comme un jeu au sens de la théorie du même nom.

Le but d'un jeu est toujours, et nécessairement, de rejoindre l'anneau W du jeu, c'est à dire de gagner un "dignus est entrare in nostro docto corpore".

Les jeux technico scientifiques sont relativement simples. Si l'auteur vise l'académie des sciences, il lui faut et il lui suffit de faire alliance avec la majorité de cette assemblée. La plupart des publications visent des jeux de ce type. De là découlent des habitudes d'exposé. Les jeux de type commerciaux ou industriels sont plus simples encore. Les entreprises tendent à conserver leurs secrets de fabrication de telle sorte que, sauf exception, leur règle serait plutôt la non publication.

Un programme est divulgué à partir du moment où il est sur le marché. Pour autant que REDACTOR soit concerné, il n'y a pas de secret que nous puissions protéger. Le jeu de "l'intelligence artificielle" est un jeu nouveau. Nous avons conçu notre stratégie dans ce jeu suivant les principes de la théorie. Il n'y a pas d'arbitre qui puisse actuellement décider du "dignus est". L'anneau W du jeu n'est pas actuellement prévisible. La seule certitude est que le jeu est planétaire et non hexagonal.

ELP est aux antipodes de ce qui se fait. En ce sens nous intervenons sur le jeu en proposant des normes différentes, des manières de faire différentes et des contraintes différentes et cette action est cohérente au regard de la théorie. ELP est aussi un produit de la théorie des jeux.

Il est notoire que la quantité de mémoire nécessaire à un certain corpus n'est pas fixe, elle a donc un minimum. Nous avons utilisé de façon arbitraire le plus petit ordinateur possible et avons ainsi défini un raffinement arbitraire de l'échelle de représentation. Ainsi, le faible encombrement des mémoires n'a pas été pour nous le souci premier, mais plutôt le moyen de raffiner notre échelle de représentation. La théorie enseigne en effet que le résultat est multiplié par la finesse de l'échelle et cette multiplication est commutative.

Nous pensons, et la théorie l'enseigne, que la demande favorisera des programmes accessibles en langage humain, tel que le français, et où l'information est fournie à l'être humain qui est ainsi en mesure de décider. Une décision récente d'une cour de justice américaine confirme notre opinion. Considéré comme un premier pas dans une stratégie à long terme, REDACTOR nous laisse espérer un milieu de partie jouable.

Le but de REDACTOR est de fabriquer des actes de société et divers documents nécessaires à la constitution de la société.

Un contrat de société peut être conçu comme une formule dont il s'agit de remplir les blancs.

A l'exception d'un seul symbole, une flèche, la conversation se déroule uniquement en français, et il n'y a pas d'autre procédure pour l'introduction des données que le français (la flèche est signifiante; elle indique qu'une question est posée ou reposée; mais elle est intuitivement compréhensible; sa grammaire est assez voisine de celle de la fameuse souris mais elle est entièrement gérée par la machine).

L'occupation en mémoire est faible, jamais plus de 31 Ks, l'ensemble de tous les overlays peut être réduit à moins de 64 Ks, la consommation de place est négligeable: 0,2 K par participant. Le programme édite un questionnaire adapté à la situation.

L'aspect conversationnel ou convivial réside dans le fait que les questions soient véritablement adaptées à la situation. Toutes les questions utiles sont posées, aucune question inutile ne l'est. Toute question question posée l'est en français et sa grammaire est strictement celle du français.

Chaque écran questionnaire est constitué ainsi:

- les lignes 6 à 28 sont réservées à l'édition d'un nombre variable de questions dont le libellé lui-même n'est pas fixe.

- ces questions correspondent au parcours d'un arbre. Le nombre maximum de branches est sept. Les niveaux de précision du parcours sont variables. Chaque question n'est définie à priori qu'en terme de structure à partir de laquelle une question peut être ou ne pas être éditée.

- le nombre de structures de questions est faible, mais les contenus des questions sont générées. On obtient ainsi un nombre fini mais assez grand de questions, ce qui permet de faire face à toutes les situations envisageables.

- le nombre total de questions peut être grossièrement évalué à 20.000 (20.000 est le nombre arrondi des combinaisons, mais certaines sont redondantes en raison de la symétrie du groupe ou mutuellement exclusives, de telle sorte que le nombre réel de combinaisons utiles peut être très inférieur à ce chiffre).

- les lignes 0 à 5 sont réservées à l'édition d'un résumé de l'état d'information du système qui a principalement pour but de rassurer l'opérateur en lui montrant que ses réponses ont été enregistrées par la machine.Eventuellement ces résumés sont des messages à part entière. Ils contiennent des informations juridiques qui auraient pû être ignorées de l'utilisateur et dont la connaissance lui est nécessaire.

- la ligne 30 est réservée aux deux messages du système:

1: "Pour continuer taper une touche"

correspond à une instruction d'attente permettant à l'opérateur de relire l'écran.

2: " n" d'attente "

indique le temps n évalué en secondes nécessaire pour l'exécution d'une instruction d'overlay et sollicite implicitement la patience de l'opérateur.

Aux écrans questionnaires correspondent éventuellement des écrans messages indiquant quelle est la réglementation applicable à la situation décrite par l'opérateur. →

Si une personne est libre de contracter des divers

points de vue considérés, aucun message n'est édité. Dans le cas contraire la loi ou le règlement qui limite cette liberté est édité. Une personne peut faire l'objet de plusieurs messages, d'un ou d'aucun suivant sa situation.

Pour exemple, un français mineur résidant aux USA aura droit à deux messages, un pour sa minorité, l'autre pour sa résidence étrangère, éventuellement cette résidence sera prise en considération dans un message relatif à la société elle-même (la loi française considère comme étrangère une société lorsque le montant des investissements des résidents étrangers, même citoyens français, dépasse un certain pourcentage du capital social).

Les temps d'opération sont courts, en moyenne quinze minutes pour l'introduction des données. Sur les quinze minutes consacrées au questionnaire, douze environ sont utilisées par l'opérateur, les overlays consommant une minute, les traitements deux minutes.

La probabilité d'erreur de REDACTOR, après contrôle, est inférieure à une sur un million d'actes donc une demi période de 500.000 actes (le nombre annuel total de SARL constituées est de 36.000). La probabilité d'erreur avant contrôle est de un pour mille.

Les techniques de contrôle sont élémentaires. Si l'on veut traiter une certaine chaîne, dont le traitement entraine la destruction, on fait autant de copies de cette chaîne qu'il est utile, de telle sorte que la chaîne initiale, ou l'une de ses copies, constitue l'élément de contrôle.

Cette méthode de programmation peut être utilisée chaque fois que la question qu'on envisage de traiter peut l'être par questionnaire, indépendamment du but de ce questionnaire et indépendamment du langage particulier utilisé, tout au moins pour les langages qui possèdent le concept d'adjectif.

Si l'on considère la puissance des théories employées, théorie des groupes et théories des jeux, le résultat fait penser à la montagne qui accouche d'une souris. La difficulté du problème peut s'exprimer ainsi: toute culture comporte des savoir faire et corrélativement des prohibitions. L'ensemble des méthodes qui ont permis à nos ancètres d'enseigner à chacun l'art de lire et d'écrire agissent comme un savoir faire et comme une prohibition. Or , ces méthodes font partie de notre culture et donc de nos préjugés, c'est pourquoi la difficulté n'est pas d'abord dans le langage mais dans nos propres préjugés.

Dans les langages d'Europe occidentale, la phrase élémentaire est de type sujet/verbe/complément et par abréviation SVC. Pour passer d'une description SVC à une description mathématiquement acceptable, C est décomposé en C1 qui correspond plus ou moins à l'accusatif et au datif latin, C2 correspond à l'ablatif et au locatif archaïque dans le même langage.

Traduit en mathématique C1 est le second terme d'une relation R dont S est le premier. C2 est relatif au domaine D où cette relation est vraie. Similairement, le verbe V est décomposé en deux parties V1 V2.

V1 dénote la relation R telle qu'elle est affirmée entre S/C1, V2 les éléments du verbe qui sont relatifs au domaine D de cette relation.

La description SVC devient dans la nouvelle écriture xRy,D. Nous négligeons D pour nous concentrer sur R. Nous avons dessiné REDACTOR sur le cas particulier, R'=u, où u dénote la mesure propre (à une translation près). L'action de REDACTOR consiste en effet à énumérer les contraintes pesant sur chaque participant, donc à mesurer leur liberté. La relation R (R="instituent une société") est transformée par ' de façon à devenir une mesure r'=u.

Lorsque la phrase exprime une mesure, il est commode d'écrire la relation xuy sous la forme plus usuelle y=u(x).

Les techniques d'I.A. sont susceptibles d'apporter de grandes simplifications dans la plupart des branches de l'activité humaine. Elles sont aussi capables de provoquer de sérieux dégats. On nous permettra donc une parenthèse.

Toutes les phrases élémentaires n'expriment pas des mesures:

"Jean est un bon mathématicien"

mesure Jean sur l'échelle contextuellement définie des mathématiques.

"Jean joue avec Jeanne dans le jardin"

ne mesure rien du tout, non plus que "Jean et Jeanne instituent une société". R=u est donc bien un cas particulier.

Les échelles sont relatives de sorte que l'identité des chaînes n'implique pas l'égalité des mesures.

"Jean joue avec Jeanne" est traduit librement de Noam CHOMSKY, "Jean est un bon mathématicien" est extrait d'un article de Jean Pierre DECLES, extrait qui traduit très librement aussi, devient en anglais:

"Johnny is a fine mathematician"

ce n'est pas le même sens, manifestement pas la même échelle si le professeur de mathématiques parle de son bon élève au père de celui-ci, ou si le professeur OPPENHEIMER parle de Johnny avec Klara VON NEUMANN.

Le cas particulier R=u semble être un cas particulier d'un sous ensemble de R dont il fait partie qui est le cas R=S des relations symétriques (xSy implique ySx). Continuons à traduire le bon mathématicien:

- dans la notation de CANTOR, "est" devient manifestement "est un élément".

- dans la notation de BOOLE (x=y.z) "est" devient "égal". "Egal" est une relation symétrique, "est un élément" ne l'est pas.

Il existe donc une opération ' telle que R'=S. C'est une opération de ce type que nous utilisons en transformant R en u puisque R dénote "constituent une société" qui n'est pas symétrique, tandis que y=u(x) implique x=u(y).

L'opération ' définie par R'=S semble jouer un rôle considérable dans le mode de pensée humain. La mesure, la logique en dépendent. La validité de ' définie par R'=u dépend de la symétrie de l'ensemble X des x, où x dénote des objets ayant même mesure.

Le raisonnement qu'on dit "magique" revient à considérer comme symétrique une relation qui ne l'est pas. Il est bon de noter que ce type de raisonnement ne fut pas le fait des sorcières auxquelles on l'impute et dont on ne sait rien, mais le fait de magistrats pétris de logique romaine.

L'ennui des raisonnements analogiques est qu'ils constituent en première analyse une symétrisation, non pas nécessairement fausse, mais qui n'est pas strictement justifiée. L'avantage des analyses fondées sur la mesure est qu'elles le sont. Dire

d'une chose qu'elle est verte ou belle, c'est la mesurer sur l'échelle des couleurs ou celle de l'esthétique. Le problème se complique un peu lorsqu'il s'agit d'opérer. Si vert est une graduation sur l'échelle des couleurs, véronèse associé à vert est un raffinement. En admettant que le nombre a dénote vert et le nombre b dénote véronèse, il doit exister une opération * telle que a*b=c, c correspondant à vert véronèse. Celà implique que le groupe a,b,c soit dense et compact.

Nous nous donnons une table ainsi conçue:

```
  1ère ligne 0       1 2 3 4 5 6 7
  2ème ligne 1       8 9 10
```

Si cette table était carrée, il suffirait d'écrire les nombres entiers les uns à la suite des autres. Si de plus cette table était décimale, on écrirait:

```
  1ère ligne 0       00 01 02 03 04 05 06 07 08 09
  2ème ligne 1       10 11 12 13 .....
```

donc on passerait d'une ligne à l'autre par une concaténation :

  1 : 3=13

: dénotant la concaténation qui est évidemment dense, compacte et non commutative.

Les échelles fournies par le langage ne sont ni carrées, ni décimales. Si au lieu d'une base dix nous acceptons l'idée d'une base variable b0 pour vert, b1 pour véronèse, en tout état de cause l'élément placé à la première colonne de la seconde ligne s'écrira 10.

Ndlr: un cas de nombres avec digits à base variable est celui des minutes et des secondes dans l'intitulé d'une valeur de temps; exemple 2h26mn35sec; dans 26mn, les unités sont en base 10, les 'dizaines' en base six.

Ainsi défini, : satisfait aux conditions fixées pour *.

Nous nous sommes posés la question de savoir si une approche non mathématique permettrait d'aboutir au même résultat. En première approximation la réponse est positive.

Dans une conception mathématique des choses, on mesure le sens de la phrase sur plusieurs axes, on pourrait aussi bien mesurer sur un seul axe, donc sur une séquence de n nombres entiers, à chacun de ces nombres correspond une action du programme. Si l'on permutte ces nombres, rien n'est changé tans qu'on permute simultanément les actions. On peut donc à partir de l'ensemble de mesure obtenir un autre enseble qui pourrait n'être ni dense, ni conpact, ni entier. Il ne serait pas même nécessaire que cet ensemble contienne au moins n. Le résultat serait conservé tant que la bijection serait maintenue.

C'est une conclusion extrèmement satisfaisante parce que conforme au sens commun. Nous parlons français tous les jours sans aucune référence mathématique si la théorie confirme que celà se peut, c'est qu'elle est conforme au sens commun, et par la suite a quelque chance d'être vraie.

Pour écrire ELP nous avons substitué à l'échelle pauvre de la langue, pauvre relativement puisque liée à l'appareil sensoriel humain, une échelle numérique tout aussi pauvre qu'elle. Ce qui nous intéressait n'était pas de mesurer précisément un phénomène mais le langage qui le décrit. Comme le nombre des graduations est faible, les nombres décrivant ces graduations sont petits, tiennent peu de place. Leur traitement prend peu de temps.

En fin de compte, nous avons pris au pied de la lettre ce mot qu'on attribue à Rutherford:

"qualitative is poor quantitative" ■

FORTH R4 R8 (Questions/Réponses)

REPONSE 4 (A. Jaccomard, 29190 PLEYBEN).

Récursion sans saturation de RP

J'ai trouvé ce code dans FORTH DIMENSIONS, volume VIII/5, Jan/Fev87, sous la signature de C. SHATTUCK. Une petite correction était nécessaire pour l'adapter à F83. Le voici :

```
: MYSELF  >IN @  ' ( mot suivant )  ['] THEN = NOT    \ la fin ?
    IF  DUP  >IN !  THEN                         \ alors rétablit le flot d'entrée.
    LAST @ NAME>                                 \ adr de compilation du mot défini
    '  ( mot suivant )  ['] ; =                  \ fin de la définition ?
    IF  COMPILE BRANCH >BODY <RESOLVE            \ alors branche au début.
    ELSE ,                                       \ sinon appel récursif à la fonction.
    THEN  >IN ! ; IMMEDIATE
```

Voici un exemple de récursion 'de queue' (appel en fin de mot) :

```
: PGCD   (S a b -- pgcd )
    ?DUP  IF  SWAP OVER MOD  MYSELF  THEN ;
```

Et un exemple de récursion 'vraie' :

```
: FACTORIELLE  (S n -- n! )
    DUP  1 = NOT  IF  DUP  1- MYSELF *  THEN ;
```

Suite page 14

UN METACOMPILATEUR SIMPLE

par Guy T. GROTKE
et
Guy M. KELLY

Résumé: en admettant un certain nombre de limitations, il est possible de bâtir un métacompilateur interactif, progressif, facile à comprendre et à utiliser.

Les limitations: le système hôte est le système cible. Le code cible doit être compilé en mémoire. Seules les références vectorisées en avant sont permises. Les systèmes hôtes et cibles partagent les mêmes variables système. Les systèmes hôtes et cibles partagent les mêmes tampons disques.

Les avantages: les mots peuvent être compilés et testés individuellement. Le code source à compiler est du pur Forth. Il ne requiert pas d'assembleur spécifique. Aucun mot particulier n'est à compiler. La métacompilation est similaire à la compilation normale. Le processus est simple à enseigner.

Historique: Lorsqu'il a été question pour un des auteurs d'enseigner Forth à l'Université de Californie, la question se posa: "Quelle version?". Un directeur du Forth Standard Team décréta la seule réponse possible: "le Forth 83-Standard!". Parfait, mais où obtenir une version du domaine public de la version 83-Standard? Qu'en était-il de la version Laxen et Perry? Une solution élégante et vraiment complète (qui soit sûre), mais assez avancée (en-têtes 'hash-code' et mots méta mixés au code source de Forth) et très anti-conformiste extérieurement (Forth en tant que fichier avec son DOS, exploitant les fichiers d'un DOS différent) facilement assimilable. La solution? Utiliser F83 comme modèle, mais en simplifiant la structure interne et en utilisant la puissance des primitives avec la puissance du "self-boot et règlant l'univers" pour approcher le matériel.

Le concept: créer deux vocabulaires successifs, META et TARGET, puis ne jamais faire de recherche de mot dans FORTH lors de la compilation dans TARGET. Partager les tampons disque et la zone des variables utilisateur entre système méta et cible. Définir une version de LITERAL qui soit translatable en mémoire. Compiler le système cible dans une seule zone mémoire séparée. Et, "à la vraie mode Forth", écrire le système source de la cible dans l'ordre bas-haut. Ceci sous-entend que la résolution des références avant ne peut être réalisée, mais permet le test d'exécution des mots avant la compilation de tout le système cible! Chaque mot devant être utilisé avant définition dans la cible doit être vectorisé (DEFER).

Le code source du métacompilateur élémentaire:

Très peu de définitions sont requises pour l'implémentation du méta-compilateur. Le code source qui suit assure que le système de base est standard par rapport à la norme 83-Standard et que l'espace mémoire est suffisant pour contenir le système cible. Pour cette raison, le système hôte et le système cible doivent résider ensemble en mémoire, car aucune table de symboles n'est générée en dehors du dictionnaire de la cible. Les définitions écrites en code machine montrées ici sont valables pour le microprocesseur 6502.

La première tâche consiste à étendre le vocabulaire assembleur de manière à contenir tous les mots utilisés dans le code source de la cible pour les mots définis en code machine. Les extensions requises sont produites pour le système cible et les mots définis en assembleur.

```
ALSO ASSEMBLER DEFINITIONS

: HEX HEX ;      : DECIMAL DECIMAL ;     : DUP DUP ;
: + + ;          : 1+ 1+ ;               : 2+ 2+ ;
: - - ;          : 1- 1- ;               : 2- 2- ;
: HERE HERE ;
```

Et on s'arrête là, car la majeure partie de la sécurité du métacompilateur est due au fait que l'ordre de recherche ne se poursuit pas dans le vocabulaire FORTH. Le résultat est que quand un mot défini exclusivement dans le vocabulaire FORTH hôte est rencontré dans le code de la cible, il se produit un message d'erreur et une définition incorrecte sera compilée. En dehors de ces quelques définitions dans le vocabulaire ASSEMBLER, il faut prendre des précautions pour s'assurer que l'ordre de recherche est respecté durant la métacompilation des définitions en code machine dans la cible.

L'étape suivante consiste à définir deux nouveaux vocabulaires:

```
ALSO <ONLY> DEFINITIONS
  VOCABULARY META
  VOCABULARY TARGET  META DEFINITIONS
```

Deux variables sont nécessaires pour la définition de la zone mémoire utilisée par le système cible:

```
VARIABLE ORIGIN   VARIABLE FINISH

: IN-TARGET? ( adr --- b  vrai si adr dans la cible)
   DUP FINISH a  U< SWAP  ORIGIN a  U< NOT AND ;
```

La partie compilation de LITERAL doit être redéfinie de manière à compiler la partie exécution d'un litéral définie dans la cible lorsqu'une valeur litérale est rencontrée dans le code source de la cible.

```
 VARIABLE  'LIT  ( prend le cfa de la procédure exécution
d'une valeur litérale)
  ' (LIT) 'LIT !  ( utilise maintenant le système hôte)

 :  <LITERAL>     (  --- redéfinition  de  la  partie
compilation)
 STATE @   IF
 DP     @  IN-TARGET? IF
 'LIT  @   ,          ELSE
 COMPILE (LIT)        THEN  ,  THEN  ;  IMMEDIATE

 ' <LITERAL> IS LITERAL ( vérifier que LITERAL est un mot
vectorisé)
```

Si le LITERAL du système hôte n'est pas vectorisé, il sera nécessaire de connecter le nouveau <LITERAL> dans l'ancienne définition. Il ne suffira pas de redéfinir simplement LITERAL.

La compilation dans deux zones mémoire différentes est réalisée en basculant la valeur du pointeur de dictionnaire DP.

```
VARIABLE T-DP    2 ALLOT   ORIGIN @  T-DP !

: TARGET-DP  ( --- bascule en zone cible)
T-DP @  DP  @  T-DP 2+ !  DP !  ;

: FORTH-DP   ( --- bascule en zone hote)
DP  @  T-DP !  T-DP 2+ @ DP ! ;
```

Plusieurs définitions sont à rajouter au vocabulaire META. Elles sont analogues aux extensions ASSEMBLER citées précédemment, mais sont utilisées comme directives de compilation de haut niveau.

```
 :  INTERPRET-ONLY  STATE @ IF BLK @ .  HERE COUNT
TYPE  TRUE ABORT" is interpret only!" THEN ;
: HOST       ONLY FORTH ;
: ESCAPE     HOST QUIT  ;  ( pour mise au point)
: DEFINITIONS DEFINITIONS ;
: (          [COMPILE] (  ;  IMMEDIATE
: [COMPILE] [COMPILE] [COMPILE] ; IMMEDIATE
: ASSEMBLER  ASSEMBLER  ;
: IMMEDIATE IMMEDIATE ;
: COMPILE-ONLY  COMPILE-ONLY  ;
: ASCII [COMPILE] ASCII ; IMMEDIATE
```

Les mots suivants sont définis tels que certains mots vectorisés dans le système cible agissent temporairement avec l'action des mots du système hôte pour permettre la mise au point d'une cible en cours de métacompilation. Ensuite, chaque mot sera restauré dans ses fonctions initialement envisagées dans le système cible.

```
: <BLOCK>  (BLOCK)  ;
: <LOAD> LOAD ;
: <CRASH> R> 2- @ >NAME  .ID
TRUE  ABORT" must have a VECTOR! <Meta>" ;
```

→

Ceci n'autorise les références avant qu'au travers des mots vectorisés.

```
: (LABEL) CREATE IMMEDIATE TARGET-DP HERE FORTH-DP ,
  DOES> @ STATE @ IF
 [COMPILE] LITERAL THEN ;

 : LABEL ( --- rend une adresse cible dans un label
hote)
 CURRENT @ ( sauvegarde de CURRENT)
 ALSO META DEFINITIONS -ALSO ( CURRENT = META)
 FORTH-DP (LABEL) TARGET-DP ( compile le label)
 CURRENT ! ; ( restaure CURRENT)
```

Ensuite on commence à définir certains paramètres du système cible.

```
 : EQU ( n --- crée une constante dans META)
   CONSTANT ; IMMEDIATE

 LIMIT    EQU   E-MEM
 FIRST    EQU   BUF1
 #BUFFERS EQU   NBUF
 RPO @    EQU   INIT-RPO
 SPO @    EQU   INIT-SPO

 HEX 0800 ORIGIN ! 4FFF FINISH ! ( exemple pour cible
basse)
 ( 5000 ORIGIN ! ABB4 FINISH ! ( exemple pour cible
haute)
 DECIMAL
```

Maintenant nous commençons à compiler le système cible. Les défintions META seront développées ultérieurement, mais les labels définis maintenant seront nécessaires plus loin. Il s'agit des points d'entrée et les valeurs par défaut des variables utilisateurs.

```
 ORIGIN a T-DP !      TARGET-DP
 ALSO ASSEMBLER

 ( cold start: )      NOP, HERE JMP,
 ( warm start: )      NOP, HERE JMP,  -ALSO

 LABEL INIT-FORTH        0 ,        ( user area + 08)
 LABEL INIT-UP        12 @ ,        (             10)
                      INIT-SPO ,    (             12)
                      INIT-RPO ,    (             14)
                      INIT-SPO 1+ , ( TIB area    16)
                      10       ,    ( BASE        18)
                      -1            ( WARNING     20)
 LABEL INIT-FENCE     0        ,    ( FENCE       22)
 LABEL INIT-DP        0        ,    ( DP          24)
 LABEL INIT-VOC-LINK     0 ,        ( VOC-LINK    26)
 LABEL UP             12 @     ,    ( UP          28)
```

<LIT> est la première définition du système cible réalisée en code machine. Une fois définie, le contenu de la variable 'LIT est initialisée avec la valeur de son cfa.

```
 ALSO ASSEMBLER

 LABEL <LIT> HERE 2+ ,       ( ceci sert de cfa)
       IP )Y LDA, PHA, IP INC, O= IF, IP 1+ INC, THEN,
       IP )Y LDA,      IP INC, O= IF, IP 1+ INC, THEN,
       <LIT> 'LIT !          ( initialise 'LIT )
```

Les labels suivants sont dépendants du système. Ces codes sont définis pour le mP 6502, mais le code peut être adapté à d'autres processeurs en changeant d'assembleur.

```
 LABEL TPUSH  DEX, DEX,

 LABEL TPUT   BOT 1+ STA, PLA, BOT STA,

 LABEL TNEXT  1 # LDY,  IP )Y LDA, W 1+ STA,
              DEY,      IP )Y LDA, W   STA,
              CLC,  IP LDA,  2 # ADC, IP STA,
              CS IF,  IP 1+ INC,  THEN,
              W 1- JMP,

 LABEL TPOPTWO INX, INX,

 LABEL TPOP   INX, INX, TNEXT JMP,

 LABEL TSETUP .A ASL,  N 1- STA,       BEGIN,
              BOT LDA, N ,Y STA,  INX, INY,
              N 1- CPY,           O=  END,
              O # LDY, RTS,
```

Les fragments de code qui suivent et les définitions sans en-tête sont toutes requises pour créer une machine virtuelle FORTH opérationnelle. Parce qu'elles sont définies dans la cible, le métacompilateur peut compiler le code source en adressage réel sans faire appel à une tablde symboles pour une résolution ultérieure d'adresse. Notez l'emploi du label (V) dans la dernière ligne de code de (DOES>). (V) est une définition META qui empile une adresse sur la pile de données pour les instructions de saut. Pour le moment concernant (V), l'ordre de recherche est ASSEMBLER, META et le vocabulaire courant est META. Si chaque code avait un en-tête, ils serait lié incorrectement dans l'ancien vocabulaire du système META même en étant déjà défini dans l'espace mémoire destiné à la cible.

```
 LABEL (DEFER) CLC, W LDA, 2 # ADC, W STA, CS IF,
               W 1+ INC,                    THEN,
               W )Y LDA, PHA, INY, W )Y LDA,
               W 1+ STA, PLA, W STA, DEY, W 1- JMP,

 LABEL (:)     IP 1+ LDA, PHA, IP LDA, PHA, CLC,
               W LDA, 2 # ADC, IP STA, TYA, W 1+ ADC,
               IP 1+ STA,  TNEXT JMP,

 LABEL (C)     2 # LDY, W )Y LDA, PHA, INY, W )Y LDA,
               TPUSH JMP,

 LABEL (V)     CLC, W LDA, 2 # ADC, PHA, W 1+ LDA,
               O # ADC, TPUSH JMP,

 LABEL (U)     CLC, 2 # LDY, W )Y LDA, UP ADC, PHA,
               INY, W )Y LDA, UP 1+ ADC, TPUSH JMP,

 LABEL (EXIT) HERE 2+  ,  ( ceci est un cfa!)

 LABEL (;)     PLA, IP STA, PLA, IP 1+ STA, TNEXT JMP,

 LABEL (DOES>) IP 1+ LDA, PHA, IP LDA, PHA,
               W )Y LDA, CLC, 3 # ADC, IP STA,
               INY, W )Y LDA, O # ADC, IP 1+ STA,
               (V) JMP,                          -ALSO
```

Maintenant que toutes les procédures ainsi définies ont une adresse réelle, tous les mots de définition peuvent être créés dans le vocabulaire META. Notez que la définition de HEADER ne génère pas de structure particulière. Si la structure des en-têtes du système cibles diffèrent de celles du système hôte, il faudra envisager quelques modifications en fonction de la structure untilisé par votre cible.

```
 FORTH-DP  HEX

 : M:  [COMPILE] :  ; IMMEDIATE
 : M;  [COMPILE] ;  ; IMMEDIATE

 M: HEADER  ( --- crée juste un nom et un en-tête lié)
       INTERPRET-ONLY
       HERE   80 C, DEFINED    IF
       HERE   WARN             THEN
       DUP 1+  Ca  1+ ALLOT
       LATEST , CURRENT @ !
       DP C@  OFF =            IF
       1 ALLOT LATEST DUP 1+
       OVER HERE SWAP - CMOVE>
       1 CURRENT @ +!          THEN M;

 M: <HEADER>  [COMPILE] HEADER M;

 M: : CURRENT @ CONTEXT !
     [COMPILE] HEADER (:) , HIDE ] M;

 M: ; (EXIT) , REVEAL [COMPILE] [ M;

 M: CREATE    [COMPILE] HEADER (V) , M;

 M: CONSTANT  [COMPILE] HEADER (C) , , M;

 M: VARIABLE  [COMPILE] HEADER (V) , O , M;

 M: USER      [COMPILE] HEADER (U) , , M;

 M: CODE      [COMPILE] HEADER HERE 2+ ,
              HIDE ALSO ASSEMBLER      M;

 M: DEFER     [COMPILE] HEADER (DEFER) ,
              CRASH ,                  M;
```

→

```
M: IS          INTERPRET-ONLY ' >BODY ! M;

M: ZERO-LINK INTERPRET-ONLY 0 , >LINK ! M;

M: 83STD      ( seulement pour votre structure d'en-têtes)
              LATEST C@ 40 OR LATEST C! M;

M: '          INTERPRET-ONLY '          M;
```

Les mots ci-dessus sont les dernières définitions de META. A partr de là, tout le code source génère des définitions TARGET. Dans notre code source cible, nous commencerons par les variables utilisateur.

```
DECIMAL ONLY META ALSO TARGET DEFINITIONS
TARGET-DP

00 USER TOS  ZERO-LINK TOS
02 USER ENTRY
04 USER LINK
06 USER SPO
08 USER RPO
10 USER TIBO
12 USER BASE  83STD
...
76 USER LAST
```

Enfin dans le code source de la cible, les mots vectorisés seront définis. Voici la liste complète des mots vectorisés de votre système. A noter que certains ne sont pas nécessaires pour les références en avant, mais sont définis pour permettrè la redirection aisée d'exécution apr l'utilisateur.

```
DEFER HEADER            ' <HEADER>    IS HEADER
DEFER LOAD    83STD     ' <LOAD>      IS LOAD
DEFER BLOCK   83STD     ' <BLOCK>     IS BLOCK
DEFER LITERAL 83STD     ' <LITERAL>   IS LITERAL
DEFER PAUSE   : NOOP ;  ' NOOP        IS PAUSE
DEFER BOOT
DEFER STATUS
DEFER SOURCE
DEFER WARN
DEFER WHERE
DEFER BUFFER 83STD
DEFER EMPTY-BUFFERS  83STD
DEFER +OFFSET
DEFER RD
DEFER WR
DEFER QUIT    83STD
DEFER KEY     83STD
DEFER KEY?
DEFER EMIT    83STD
DEFER CR      83STD
DEFER PAGE
```

Le reste du code source cible sont les constantes, variables, définitions en code machine et définitions "deux-points". Lors de la compilation d'un mot vectorisé, le vecteur utilisé est celui défini par IS.

```
: (NUMBER)     ( adr --- D)
   NUMBER? O= ?MISSING ;

' (NUMBER) IS NUMBER
```

Le listing complet ne figure pas ici, mais a été essayé par l'auteur, et vous êtes assurés qu'il reproduit exactement le FORTH écrit en FORTH.

Après la définition des mots COLD et WARM, les parties définies en code machine commençant leur exécution seront compilées, et les pointeurs correspondants connectés au début du système cible.

Si vous rencontrez des difficultés lors de la métacompilation, il suffira de taper ALSO FORTH et toutes les facilités normales du système hôte seront utilisables pour la mise au point. Vous pouvez DUMPer le code compilé, vérifier les noms et les lister avec WORDS, puis essayer les mots de la cible pour vérifier leur bon fonctionnement. Pour revenir à la métacompilation, il faut taper -ALSO et poursuivre le chargement. Lorsque la cible sera terminée, activez sont interpréteur. A partir de ce moment, pour revenir au système hôte, il faut faire appel à une "clause d'echappement", nommée justement ESCAPE. Dans le code source cible, nous avons défini une table d'exécution des caractères et une routine chargée d'entrée dans EXPECT ou d'exécuter une séquence selon le caractère entré au clavier. Un de ces codes exécute la séquence ESCAPE.

```
: ESC-IN ( c --- action à valider pour touche ESCAPE)
  ESCAPE  ( définition META)
 ABORT" Reset " ;
```

A la fin du code source de la cible, les paramètres de démarrage doivent être initialisés.

```
LATEST INIT-FORTH      ! ( LATEST  )
HERE   INIT-UP 12 +    ! ( FENCE   )
HERE   INIT-DP 14 +    ! ( DP      )
VOC-LINK @ INIT-DP 16 + ! ( VOC-LINK)

: START  ( saute sur le point d'entrée à froid)
0 +ORIGIN GO ;
```

Pour activer le système cible, taper START.

Extension et modification du métacompilateur:

La version originale de ce métacompilateur a été écrite à partir d'un système FigFORTH tournant sur IBM PC. Le système Fig a été arrangé de manière à vectoriser les mots appropriés, le concept ONLY créé par RAGSDALE dans le vocabulaire ALSO (Fort 83 Standard, section c, proposition expérimentale "spécification de l'ordre de recherche et contrôle") a été rajouté. Ce métacompilateur est capable de générer un système F83 avec des structures d'en-tête différentes de celle du système hôte ce qui requiert la redéfinition des versions Fig de CREATE et -FIND. La modification des structures d'en-tête pour en-têtes séparés ne requiert pas d'effort supplémentaire.

L'utilisation du métacompilateur pour créer un système cible destiné à un autre type de microprocesseur requiert l'utilisation d'un assembleur et quelques mots pour reloger l'image système dans l'ordinateur cible. Cependant, les outils de mise au point ne peuvent être utilisés car ce système cible ne peut fonctionner sur le système hôte.

Conclusion:

La simplicité de ce métacompilateur réside dans ses facilités d'adaptation et d'extension sur votre propre système FORTH.

Note de la rédaction:

Les personnes ayant le forth 83-Standard de Laxen et Perry (version CP/M ou MSDOS) disposent déjà d'un métacompilateur très performant. Pour implanter une version personnalisée sur le système de leur choix, il faut modifier le contenu du fichier META80.BLK pour la version F83 CP/M, des fichiers KERNEL86.BLK et META86.BLK pour la version MSDOS. L'assembleur se situe dans les fichiers CPU8080.BLK sous CP/M et CPU8086.BLK sous MSDOS.

Nous espérons vous donner prochainement des précisions plus détaillées sur l'emploi du métacompilateur F83 de Laxen et Perry.



```
: clc  ( bit de carry:=0)
ASSEMBLER  FE # andcc  FORTH  ;

: sec  ( bit de carry:=1)
 ASSEMBLER  1 # orcc  FORTH  ;  DECIMAL
```

L'assemblage conditionnel n'est possible qu'au sein d'une macro-instruction, mais peut par contre faire appel à toutes les structures de contrôle existant en FORTH, telles BEGIN..UNTIL, CASE..OF..ENDOF..ENDCASE, etc... (attention, ne pas confondre if et IF; voir LABELS ET STRUCTURES DE CONTROLE).

DEFINITION DE SOUS-PROGRAMMES

Cette méthode reprend le principe expliqué dans le manuel de référence du FORTH THOMSON, à la différence que le vocabulaire ASSEMBLER doit être cité explicitement avant l'utilisation des mnémoniques d'assemblage. Une séquence d'assemblage sera terminée par rts. Schéma de définition:

```
CREATE <mot>  ASSEMBLER
..définition en assembleur.. rts FORTH
```

L'exécution du sous-programme, en FORTH, est réalisée par la séquence <mot> CALL. L'appel de la même définition, en tant que sous-programme à partir d'une autre définition en cours d'assemblage est réalisée par la séquence <mot> jsr (ou bsr).

Si le sous-programme est destiné à n'être utilisé qu'au sein d'une autre définition écrite en assembleur, on peut créer un sous-programme n'ayant pas d'en-tête dans le dictionnaire FORTH. Schéma de définition:

```
ASSEMBLER  LABEL <label1>
      ..définition en assembleur.. rts FORTH
```

Et plus loin, dans la définition utilisant le sous-programme commençant à l'adresse pointée par <label1>, on retrouvera la séquence d'assemblage suivante:

```
.. <label1> jsr  ( ou bsr)
FORGET-LABEL  <label1> ( si label1 plus utilisé  par  la
suite) ..
```

LES PROGRAMMES

On entend par 'programme', tout mot dont l'exécution est similaire aux primitives du dictionnaire déjà définies en assembleur. De tels mots sont définis par l'assembleur FORTH à l'aide des mots de définition CODE et END-CODE. Ces mots sont à une définition écrite en assembleur, ce que sont les mots ':' (deux-points) et ';' (points-virgule) pour une définition écrite en langage FORTH. Schéma de définition:

```
CODE <mot>
.. définition en assembleur
END-CODE
```

Toute commande FORTH située entre CODE et END-CODE sera immédiatement exécutée, que ces mots FORTH soient d'exécution immédiate ou non.

Un mot défini par CODE..END-CODE n'est exécutable qu'à partir de FORTH. Il ne peut servir simultanément de sous-programme dans une autre définition écrite en assembleur. Si vous désirez qu'une définition puisse être exécutable simultanément depuis FORTH ou en tant que sous-programme dans une autre définition assembleur, il faut définir deux mots séparés, le premier étant un sous-programme utilisé par le second. Schéma de définition:

```
CREATE <mot1>  ASSEMBLEUR
.. définition en assembleur...  rts FORTH

CODE <mot2>
..  <mot1> jsr  .. NEXT END-CODE
```

Par convention les mots <mot1> et <mot2> auront le même libellé, mais il est conseillé d'écrire le libellé de <mot1> entre parenthèses afin d'indiquer qu'il s'agit d'une primitive à ne pas utiliser directement (un peu comme DO et (DO) dans le dictionnaire FORTH).

Mais la possibilité d'utiliser un mot aussi bien depuis FORTH que comme sous-programme assembleur a été envisagée. La méthode utilisée n'est pas standard, car assez personnelle. Elle consiste à dériver l'adresse contenue dans le cfa du mot défini par CODE..END-CODE vers une adresse autre que le début de la zone paramétrique du mot défini. Schéma de définition:

```
CODE <mot>
 .. déf. en ass... rts
HERE>CFA  NEXT  END-CODE
```

Le mot <mot> peut dorénavant être utilisé en tant que sous-programme dans une autre définition assembleur:

```
... FIND <mot> 2+ jsr ...
```

Le mot HERE>CFA utilisé dans la définition de <mot> est une macro-instruction assemblant un branchement relatif vers l'adresse du début de la zone paramétrique du mot en cours d'assemblage et génère une indirection dans le cfa de ce même mot. Le mot <mot> peut être exécuté en FORTH en tapant simplement <mot> et précédé des éventuels paramètres utilisés par <mot>.

LES DEFINITIONS MIXTES

A partir de cet instant, accrochez-vous aux branches, asseyez-vous et n'oubliez pas de respirer, car ce que vous allez lire est tout à fait révolutionnaire. En effet, il s'agit de définir des mots FORTH dont certaines parties sont écrites et exécutables en FORTH et d'autres parties définies en assembleur et exécutables en code machine. Une telle possibilité n'existe, à ma connaissance, sur aucun autre langage informatique.

Pour ce faire, nous disposons de deux mots, ASM[et]FORTH. Ces deux mots sont utilisés dans une définition "deux-points" et encadrent une séquence écrite en assembleur. Schéma de définition:

```
: <mot>
..déf.en FORTH..
ASM[   ...définition en assembleur.. NEXT
]FORTH ...définition en FORTH ... etc...  ;
```

On peut passer autant de fois qu'on le souhaite de FORTH à l'assembleur et inversement. Toute séquence écrite en assembleur doit être encadrée par ASM[et]FORTH et se termine par la macro-instruction NEXT.

LISTING

```
SCR: 7
( ASSEMBLER PAGE 01/..              MP21jan86)

: ?INIT ( ---)
CR ." Vecteur non initialise " 3 ERROR ;

: DEFER
CREATE [ FIND ?INIT ] LITERAL ,
DOES>  @ EXECUTE ;

: [']  ( ---)
FIND  [COMPILE] LITERAL ; IMMEDIATE

: IS      ( cfa ---)
FIND 2+ STATE @
IF    [COMPILE] LITERAL COMPILE !
ELSE  !              THEN ; IMMEDIATE

;S Creation de mots vectorises au for-
mat 83-STANDARD. Exemple:

DEFER TEST        cree un en-tete TEST
['] CLS IS TEST   affecte le CFA de CLS
                  au PFA de TEST. Dorena-
                  vant, TEST execute CLS
```

```
SCR: 8
( ASSEMBLER PAGE 02/..          MP21jan86)

VOCABULARY ASSEMBLER IMMEDIATE
      ASSEMBLER DEFINITIONS
DEFER LITERAL  IMMEDIATE
  FORTH FIND LITERAL
                 ASSEMBLER IS LITERAL
DEFER C,
  FORTH FIND C,    ASSEMBLER IS C,
DEFER ,
  FORTH FIND ,     ASSEMBLER IS ,
DEFER HERE
  FORTH FIND HERE  ASSEMBLER IS HERE
DEFER ALLOT
  FORTH FIND ALLOT ASSEMBLER IS ALLOT
DEFER !
  FORTH FIND !     ASSEMBLER IS !
DEFER C!
  FORTH FIND C!    ASSEMBLER IS C!
DEFER @
  FORTH FIND @     ASSEMBLER IS @
DEFER C@
  FORTH FIND C@    ASSEMBLER IS C@
```

```
SCR: 9
( ASSEMBLER PAGE 03/..          MP11dec85)
FORTH    DEFINITIONS

: CODE
CREATE  ( --- <mot> en compilation)
HERE DUP 2- !
[COMPILE] ASSEMBLER    ; IMMEDIATE

: END-CODE  ( ---)
CURRENT @  CONTEXT ! ;

: ;CODE
?CSP COMPILE (;CODE)
[COMPILE] [
[COMPILE] ASSEMBLER
[COMPILE] SMUDGE ; IMMEDIATE ;S

 Creation d'un mot de def.en assembleur:

CODE <mot> ... END-CODE
Creation d'un mot de definition, avec la
partie execution def. en assembleur:

: <mot> ... ;CODE ... END-CODE
```

```
SCR: 10
( ASSEMBLER PAGE 04/..          MP11dec85)

ASSEMBLER DEFINITIONS

VARIABLE MODE ( precise mode adressage)
( 0=mode etendu   "espace" ou signe > )
( 1=mode immediat signe #             )
( 2=mode direct   signe <             )
( 3=mode indexe
          signe ,--R ,-R ,R ,R+ ,R++ )

: 0MODE!  ( ---)
0 MODE ! ;

: ERROR-MODE ( ---)
CR ." Mode d'adressage "
MODE @  CASE
0 OF ." etendu"     ENDOF
1 OF ." immediat"   ENDOF
2 OF ." direct"     ENDOF
3 OF ." indexe"     ENDOF
ENDCASE ."  interdit"
0MODE! 3 ERROR ;
```

```
SCR: 11
( ASSEMBLER PAGE 05/..          MP11dec85)
HEX
: INH     ( Creation des mots inherents)
CREATE ,  ( c --- <mot> en compilation)
DOES>  ( --- en execution)
MODE @  IF ERROR-MODE THEN
  @ C,  0MODE! ;

3A INH abx    48 INH asla   58 INH aslb
47 INH asra   58 INH asrb   4F INH clra
5F INH clrb   43 INH coma   53 INH comb
19 INH daa    4A INH deca   5A INH decb
4C INH inca   5C INH incb   48 INH lsla
58 INH lslb   44 INH lsra   54 INH lsrb
3D INH mul    40 INH nega   50 INH negb
12 INH nop    49 INH rola   59 INH rolb
46 INH rora   56 INH rorb   3B INH rti
39 INH rts    1D INH sex    3F INH swi
13 INH sync   4D INH tsta   5D INH tstb

: (10,) 10 C, ;    : (11,) 11 C, ;

: swi2 (10,) swi ;
: swi3 (11,) swi ;
DECIMAL
```

```
SCR: 12
( ASSEMBLER PAGE 06/..          MP11dec85)

HEX  VARIABLE MANIPULATION?

: REGISTRE
CREATE ( n1 n2 --- <mot> en compilation)
  C, C,
DOES>  ( --- adr en execution)
  >R
MANIPULATION? @  0= ( a-t-on appele un)
IF                  ( registre ?      )
  1 MANIPULATION? ! ( non=> on met 0 0)
  0 0               ( sur la pile     )
THEN
R> DUP >R
C@ + SWAP DUP  10 <
IF
  10 *  ( 16 * en decimal)
THEN
R> 1+
C@ + SWAP ;

DECIMAL
```

```
SCR: 13
( ASSEMBLER PAGE 07/..          MP11dec85)
HEX
 08 02 REGISTRE a    09 04 REGISTRE b
 00 06 REGISTRE d    01 10 REGISTRE x
 02 20 REGISTRE y    03 40 REGISTRE u
 04 40 REGISTRE s    05 80 REGISTRE pcr
 0B 08 REGISTRE dpr  0A 01 REGISTRE ccr

: PS-PL  CREATE C, ( c --- <mot> )
DOES>   MODE @ IF ERROR-MODE THEN
 C@ C, C, DROP
0 MANIPULATION? ! 0MODE! ;

 34 PS-PL pshs        36 PS-PL pshu
 35 PS-PL puls        37 PS-PL pulu

: TF-EX  CREATE C, ( c --- <mot> )
DOES>   MODE @ IF ERROR-MODE THEN
 C@ C, DROP C,
0 MANIPULATION? ! 0MODE! ;

 1F TF-EX tfr     1E TF-EX exg
DECIMAL
```

```
SCR: 14
( ASSEMBLER PAGE 08/..          MP11dec85)
HEX VARIABLE POSTCODE
    VARIABLE POSTOCTET

: SELECTION
CREATE  .C, ( c --- <mot> en compil.)
DOES>    C@ MODE ! ;

0 SELECTION >        1 SELECTION #
2 SELECTION <        3 SELECTION ,R

: (0B,)   ( ---)  ;

: (8B,)   ( ---)
POSTOCTET @ C, ;

: (16B,)  ( ---)
POSTOCTET @  , ;

DEFER DEPLACEMENT

: TRT-0BITS   ( ---)
84 POSTCODE +!
['] (0B,) IS DEPLACEMENT ;    DECIMAL
```

```
SCR: 15
( ASSEMBLER PAGE 09/..          MP11dec85)
HEX

: IND
CREATE   C,
DOES>  C@  POSTCODE !
['] (0B,) IS DEPLACEMENT  ,R  ;

80 IND ,x+     81 IND ,x++
82 IND ,-x     83 IND ,--x

A0 IND ,y+     A1 IND ,y++
A2 IND ,-y     A3 IND ,--y

C0 IND ,u+     C1 IND ,u++
C2 IND ,-u     C3 IND ,--u

E0 IND ,s+     E1 IND ,s++
E2 IND ,-s     E3 IND ,--s

DECIMAL
```

```
SCR: 16
( ASSEMBLER PAGE 10/..          MP27dec85)

HEX

86 IND a,x     85 IND b,x     8B IND d,x
A6 IND a,y     A5 IND b,y     AB IND d,y
C6 IND a,u     C5 IND b,u     CB IND d,u
E6 IND a,s     E5 IND b,s     EB IND d,s

DECIMAL
```

```
SCR: 17
( ASSEMBLER PAGE 11/..          MP27dec85)

HEX

: TRT-4BITS   ( n ---)
DUP 0<
IF  10 POSTCODE +!  THEN
0F AND POSTCODE +!
['] (0B,) IS DEPLACEMENT ;

: TRT-8BITS   ( n ---)
88 POSTCODE +!
FF AND POSTOCTET !
['] (8B,) IS DEPLACEMENT ;

: TRT-16BITS   ( n ---)
89 POSTCODE +!
POSTOCTET !
['] (16B,) IS DEPLACEMENT ;

DECIMAL
```

```
SCR: 18
( ASSEMBLER PAGE 12/..          MP27dec85)
HEX
: ]      ( ---)
MODE @  DUP 1 = OVER 2 = OR
IF  ERROR-MODE THEN
3 =
IF                     ( si mode indexe)
  POSTCODE @
  80 AND 0=            ( si b7=0)
  IF
    POSTCODE @ DUP 0F AND SWAP 10 AND
    IF  80 +  THEN
    POSTOCTET !
    POSTCODE @ 60 AND  98 + POSTCODE !
    ['] (8B,) IS DEPLACEMENT
  ELSE
    10 POSTCODE +!
  THEN
ELSE
  9F POSTCODE !  3 MODE !
  ['] (16B,) IS DEPLACEMENT
THEN ;
DECIMAL
```

```
SCR: 19
( ASSEMBLER PAGE 13/..          MP27dec85)
HEX

: SET-DEPLACEMENT   ( n ---)
?DUP
IF
  DUP -80 FORTH < ASSEMBLER
  OVER 7F FORTH > ASSEMBLER OR
  IF    TRT-16BITS
  ELSE
    DUP -10 FORTH < ASSEMBLER
    OVER 0F FORTH > ASSEMBLER OR
    IF     TRT-8BITS
    ELSE  TRT-4BITS   THEN
  THEN
ELSE   TRT-0BITS  THEN ;

: INDN     ( n --- <mot> en compil.)
CREATE C,    DOES> C@
POSTCODE !  SET-DEPLACEMENT  ,R  ;

00 INDN ,x      20 INDN ,y
40 INDN ,u      60 INDN ,s

DECIMAL
```

```
SCR: 20
( ASSEMBLER PAGE 14/..          MP27dec85)
HEX

: RELATIF ( adr1 adr2 --- rel f1)
-           ( rel=adr1-adr2        )
DUP -80  FORTH < ASSEMBLER
OVER 7F  FORTH > ASSEMBLER OR
IF     1            ( f1=1 si -80<rel>7F)
ELSE  0  THEN  ;  ( f1=0 cas contraire)

: ,pcr  ( n ---)
HERE 2+ RELATIF
IF     2-  POSTOCTET !
       9C POSTCODE !
       ['] (16B,) IS DEPLACEMENT
ELSE  1-  FF AND POSTOCTET !
       8C POSTCODE !
       ['] (8B,) IS DEPLACEMENT
THEN ,R ;

DECIMAL
```

```
SCR: 21
( ASSEMBLER PAGE 15/..          MP27dec85)

HEX
: REG
CREATE  100 * + ,  ( c1 c2 --- <mot>)
DOES>      ( c1=1, operateur 8 bits )
MODE @     ( c1=2, operateur 16 bits)
CASE       ( c2=code operateur        )
     ( Test du mode adress. immediat   )
  1 OF  DUP 1+ C@ 1-
        IF    C@ C,  ,
        ELSE  C@ C, C,  THEN     ENDOF
     ( Test du mode adress. etendu     )
  0 OF  C@ 30 +  C,  ,           ENDOF
     ( Test du mode adress. direct     )
  2 OF  C@ 10 +  C,   100 MOD C, ENDOF
     ( Test du mode adress. indexe     )
  3 OF  C@ 20 +  C,
     POSTCODE @ C,  DEPLACEMENT ENDOF
ENDCASE
0MODE!
0 POSTOCTET !
0 POSTCODE !  ;  DECIMAL
```

```
SCR: 22
( ASSEMBLER PAGE 16/..          MP27dec85)

HEX

1 89 REG adca   1 C9 REG adcb
1 8B REG adda   1 CB REG addb
1 84 REG anda   1 C4 REG andb
1 85 REG bita   1 C5 REG bitb
1 81 REG cmpa   1 C1 REG cmpb
1 88 REG eora   1 C8 REG eorb
1 86 REG lda    1 C6 REG ldb
1 8A REG ora    1 CA REG orb
1 82 REG sbca   1 C2 REG sbcb

: NIREG
MODE @ 1 = IF ERROR-MODE THEN
REG ;

1 87 NIREG sta 1 C7 NIREG stb
 ( Ne doivent pas etre utilises en )
 ( adressage immediat              )

1 80 REG suba   1 C0 REG subb

DECIMAL
```

```
SCR: 23
( ASSEMBLER PAGE 17/..          MP27dec85)

HEX

2 C3 REG addd
2 83 REG (cmpd)     : cmpd (10,) (cmpd) ;
2 8C REG (cmps)     : cmps (10,) (cmps) ;
2 83 REG (cmpu)     : cmpu (11,) (cmpu) ;
2 8C REG cmpx
2 8C REG (cmpy)     : cmpy (10,) (cmpy) ;
2 CC REG ldd
2 CE REG ldu
2 CE REG (lds)      : lds  (10,) (lds)  ;
2 8E REG ldx
2 8E REG (ldy)      : ldy  (10,) (ldy)  ;
2 CD NIREG std
2 CF NIREG (sts)    : sts  (10,) (sts)  ;
2 CF NIREG stu
2 8F NIREG stx
2 8F NIREG (sty)    : sty  (10,) (sty)  ;
2 83 REG subd

2 8D NIREG jsr
DECIMAL
```

```
SCR: 24
( ASSEMBLER PAGE 18/..          MP28dec85)

HEX

: MEM
CREATE C, ( c --- <mot> avec c=val en )
DOES> C@  MODE @    ( adressage direct)
DUP 1 = IF ERROR-MODE THEN
CASE
      ( Test du mode adress. etendu    )
  0 OF    70 + C, ,              ENDOF
      ( Test du mode adress. direct    )
  2 OF  C, 100 MOD C,            ENDOF
      ( Test du mode adress. indexe    )
  3 OF    60 + C,
    POSTCODE @ C, DEPLACEMENT    ENDOF
ENDCASE
0MODE!
0 POSTOCTET !
0 POSTCODE ! ;

DECIMAL
```

```
SCR: 25
( ASSEMBLER PAGE 19/..          MP28dec85)

HEX

08 MEM asl          07 MEM asr
0F MEM clr          03 MEM com
0A MEM dec          0C MEM inc
08 MEM lsl          04 MEM lsr
00 MEM neg          09 MEM rol
06 MEM ror          0D MEM tst
0E MEM jmp

: IMM  ( c --- <mot> en compilation)
CREATE C,
DOES> C@ MODE @ 1 =
IF     C, C, 0MODE!
ELSE  ERROR-MODE
THEN ;

1C IMM andcc  1A IMM orcc  3C IMM cwai

DECIMAL
```

```
SCR: 26
( ASSEMBLER PAGE 20/..          MP04jan86)

HEX
: INDEX
CREATE
  C,
DOES>
  C@ MODE @ 3 =
IF     C, 0MODE!
       POSTCODE @ C, DEPLACEMENT
ELSE  ERROR-MODE  THEN  ;

32 INDEX leas
33 INDEX leau
30 INDEX leax
31 INDEX leay

DECIMAL
```

```
SCR: 27
( ASSEMBLER PAGE 21/26          MP04jan86)

HEX

: bra  ( adr ---)
HERE 1+ RELATIF
IF     2- 16 C, ,
ELSE  1- 20 C, C,  THEN ;

: bsr  ( adr ---)
HERE 1+ RELATIF
IF     2- 17 C, ,
ELSE  1- 8D C, C,  THEN ;

DECIMAL
```

```
SCR: 28
( ASSEMBLER PAGE 22/26          MP04jan86)

HEX

: RELBR
CREATE  C,
DOES>
C@ SWAP
HERE 1+ RELATIF
IF     2- (10,) SWAP C, ,
ELSE  1-  SWAP C, C,  THEN ;

24 RELBR bcc    25 RELBR bcs
27 RELBR beq    2C RELBR bge
2E RELBR bgt    22 RELBR bhi
24 RELBR bhs    2F RELBR ble
25 RELBR blo    23 RELBR bls
2D RELBR blt    2B RELBR bmi
26 RELBR bne    2A RELBR bpl
21 RELBR brn ( sans effet => 2x NOP)
28 RELBR bvc    29 RELBR bvs

DECIMAL
```

```
SCR: 29
( ASSEMBLER PAGE 23/26          MP04jan86)
HEX
: CONDITION
CREATE C,
DOES> C@ ;

24 CONDITION cc     25 CONDITION cs
27 CONDITION eq     2C CONDITION ge
2E CONDITION gt     22 CONDITION hi
24 CONDITION hs     2F CONDITION le
25 CONDITION lo     23 CONDITION ls
2D CONDITION lt     2B CONDITION mi
26 CONDITION ne     2A CONDITION pl
28 CONDITION vc     29 CONDITION vs

24 CONDITION c=0    25 CONDITION c=1
27 CONDITION z=1    2C CONDITION 0>=
2E CONDITION 0>     22 CONDITION >
24 CONDITION >=     2F CONDITION 0<=
25 CONDITION <      23 CONDITION <=
2D CONDITION 0<     2B CONDITION mi
26 CONDITION z=0    2A CONDITION pl
28 CONDITION v=0    29 CONDITION v=1
DECIMAL
```

```
SCR: 30
( ASSEMBLER PAGE 24/26          MP04jan86)
HEX
: if  ( cond --- adr)
1 XOR C, HERE  0 C, ;

: then ( adr ---)
HERE OVER RELATIF
IF    ." Branchement trop long"  3 ERROR
THEN    1- SWAP C! ;

: else  ( adr1 --- adr2)
20 if SWAP then ;

: begin  ( adr ---)  HERE ;

: while  ( cond --- adr)  if ;

: repeat ( adr1 adr2 ---)
>R bra  R> then ;

: until  ( adr cond ---)
>R HERE  1+ RELATIF
IF       2- (10,) R> C, ,
ELSE     1- R> C, C,    THEN ;
DECIMAL
```

```
SCR: 31
( ASSEMBLER PAGE 25/26          MP04jan86)
HEX
: NEXT ( macro pour TO7, TO7-70, TO9)
      ASSEMBLER  0099 jmp FORTH ;
( ou pour MO5, GOUPIL 3, VEGAS, TAVERN.)
( ASSEMBLER ,y++ ldx 0 ,x ] jmp FORTH  )
DECIMAL

: FORGET-LABEL  ( --- <nom>)
[COMPILE] ' NFA
[ HERE 40 + FORTH ] LITERAL ASSEMBLER
OVER OVER 1+ U<
IF      CR ." Erreur de label" 3 ERROR
ELSE    OVER SWAP 2 - ! PFA LFA @
        CONTEXT @ !                THEN ;

2 ALLOT
: LABEL  ( --- <nom>)
CURRENT @ HERE DEFINITIONS
[ LATEST 2 - FORTH ] LITERAL ASSEMBLER
DUP @ DP ! OVER  CONSTANT HERE SWAP !
 DP ! CURRENT ! ;

HERE LATEST 2 - ! 200 ALLOT
FORTH DEFINITIONS
```

```
SCR: 32
( ASSEMBLER PAGE 26/26                   )
( EXTENSIONS FORTH/ASSEMBL.     MP06mai86)

: HERE>CFA  ( ---)
HERE LATEST PFA CFA !
LATEST PFA ASSEMBLER jsr FORTH ;

: ASM[
BASE @ CONTEXT @
HERE 6 + ,
COMPILE BRANCH HERE 0 ,
HERE 2+ ,
[COMPILE] ASSEMBLER
[COMPILE] [ ; IMMEDIATE

: ]FORTH
HERE OVER - SWAP !
CONTEXT ! BASE !
[COMPILE] ] ;
```

REPONSE 8 (A. Jaccomard, 29190 PLEYBEN).

Compilation de fichiers texte.

Encore FORTH DIMENSIONS, volume VIII/5, Jan/Fev87, par C. A. WENRICH, article 'SCREENLESS FORTH', tout un programme. Voici le code :

```
ONLY  FORTH ALSO  DEFINITIONS  WARNING OFF
: NLOAD  .S  (LOAD) ;  ' NLOAD IS LOAD

VOCABULARY UNSCREEN  ONLY FORTH ALSO DOS ALSO UNSCREEN DEFINITIONS
VARIABLE  B/FILE   SP0 @  4096 256 + - CONSTANT MODBUF          \ tampon.

( MODBUF est placé ici à 4096+256 octets sous la pile données. Dans l'article original, l'auteur le place en
    61440 ou F000H, c-à-d qu'il utilise les 4 buffers disque )

: REC-SIZE   (S adr -- adr' )  14 + ;              \ adr taille enregist. ds FCB
: FILE-SIZE  (S adr -- adr' )  16 + ;              \ adr taille fichier ds FCB
: OPEN-FILE  (S -- )  IN-FILE @  DUP  15 BDOS  DOS-ERR?
    ABORT" Erreur d'ouverture"  FILE-SIZE @  DUP  4096 >
    ABORT" Taille > 4K"   B/FILE ! ;
: READ-CHAR  IN-FILE @  20 BDOS  DOS-ERR?
          ABORT" Erreur de lecture" ;
: READ-SEQ  (S -- )  IN-FILE @  DUP  REC-SIZE  1 SWAP !
   FILE-SIZE @ 0    DO
   READ-CHAR  PAD C@  BL  MAX  MOD-BUF  >IN @ +  C!  1 >IN +!
  LOOP ;
: (LOAD)  (S -- )  ?DEFINE  !FILES   OPEN-FILE  >IN  OFF
    PAD  SET-DMA  READ-SEQ  >IN OFF  BLK ON  RUN ;
: (SOURCE)  (S -- adr lgr )  BLK @
  IF  MOD-BUF  B/FILE @  ELSE  TIB  #TIB @  THEN ;

: (?ERROR)  (S adr lgr f -- )
  IF  TYPE  CR  SP0 @ SP!  PRINTING OFF   BLK @
     IF  CR  MOD-BUF  >IN @  BOUNDS   DO  I C@  EMIT  LOOP
     THEN  QUIT
ELSE  2DROP  THEN ;

 ' (LOAD) IS LOAD  ' (?ERROR) IS ?ERROR  ' (SOURCE) IS SOURCE
```

N'importe quel éditeur peut être utilisé, y compris EDLIN. L'auteur a redéfini les 4 tampons disque du haut de mémoire pour en faire un seul tampon de fichier source de 4K (NdT: je préfère les laisser intacts, et utiliser un autre emplacement mémoire). Tout programme plus grand que cette limite doit être fractionné en modules de 4K, puis chaînés.

Un vocabulaire, UNSCREEN, est d'abord créé.

B/FILE est une variable contenant le nombre d'octets du fichier source.

MOD-BUF est l'adresse du tampon, où ira le fichier.

REC-SIZE et FILE-SIZE sont les déplacements dans le FCB donnant les adresses de la taille d'un enregistrement et du fichier.

OPEN-FILE est semblable à la commande existante, sauf qu'ici elle vérifie que le source ne dépasse pas 4K, sinon ABORT et message d'erreur.

READ-CHAR lit un caractère du fichier source. READ-SEQ lit une séquence de celui-ci, et la place à MOD-BUF. L'adresse de transfert (la DTA) est placée en PAD. Tout caractère est comparé à BL, les 'imprimables' sont placés dans MOD-BUF, les autres remplacés par un 'blanc'.

(LOAD) active l'interprète après lecture du fichier : il combine les fonctions des commandes OPEN et (LOAD). Une fois LOAD revectorisée sur la version UNSCREEN de (LOAD), il suffit d'entrer "LOAD <nom-de-fichier.ext>" et le fichier sera ouvert, lu en mémoire et interprété.

Si aucune erreur n'est rencontrée, le "ok" familier est affiché.

Il faut revectoriser LOAD, SOURCE et ?ERROR sur le vocabulaire FORTH si vous devez utiliser à nouveau les écrans classiques.

Suite page 17

fBASEII en FORTH 83-Standard
pour ORIC ATMOS

par Jean-Luc SIRET

Puisque JEDI demande des programmes, j'envoie celui-ci sans prétention. Je l'ai pompeusement appelé FBASE2 puisque c'est un développement à partir du FBASE1 paru dans le JEDI no 22. Bien sûr pour ceux qui travaillent sur PC, il n'a pas la prétention de se comparer à l'autre D.... vraiment pas! Si vous voulez, appelez celui-ci FBASE1.007! Il a été conçu sur ATMOS avec le F83 écrit et adapté par M.ZUPAN pour cette machine.

Pour les explications premières voir le JEDI no22
En - moins de contrôle d'entrée des nombres.
En + possibilité d'une recherche sélective à l'aide de clés ou de masque.

Première modification: le nombre de paramètres de chaque champ est augmenté de 1: c'est le dernier chiffre qui indique si ce champ pourra servir de clé (1) ou non (0). Le nombre de clés maximum est limité ici à 10, mais peut être modifié facilement après le CREATE de l'écran 1 et avec les deux 10 dans TBCLE du 2nd. Le PBUILD de déclaration de champ compile maintenant l'adresse du descriptif de chaque champ, ce qui permet une recherche plus rapide.

Le principe: les clés sont mises dans des blocks mais transférées en mémoire vive au début de chaque session. Le mot PEND les sauvegarde à la fin et seulement à la fin. Donc le faire obligatoirement si le programme s'arrête pour une raison ou une autre avant.

En exemple: la taille du fichier doit être prévue avant (au moisn sur ATMOS) et surtout le nombre de block qui vont recevoir les clés. Exemple:

120 fiches de 138 octets de longueur avec 8 clés par fiche nécessiteront au total 21 blocks (1024)

le block 1: définition des champs

blocks 4 au 21: contiennent les fiches proprement dites; 7 fiches par block (pas de troncature)

block 2 et 3 pour les clés: 2 octets par clés donnent: 2x8x120=1920 octets

Dans cet exemple on écrirait 2 2 138 IREC

Le premier 2 donne le premier 'block' des clés
Le second 2 donne le nombre de 'block' de clés
La somme de ces nombres (4 ici) donne le premier block de fiches
Les clés et les fiches sont donc consécutives

Les tris: les clés n'en sont pas au sens de dBASE car elles ne codent pas tout un champ, mais seulement les 3 premiers caractères du champ alphanumérique ou la partie basse numérique. La recherche s'effectue selon ce critère. Exemple:

la clé 'jed' codera 'jedi' aussi bien que 'jedu bon tabac dans...'

Après la recherche, il y aura ici 2 fiches sélectionnées et toutes les autres recherches s'effectueront alors sur ces deux là.

Le tri par masque ne s'effectue que sur les rubriques alphanumériques. Il y a possibilité bien sûr d'utiliser un joker qui est le '?'. Evidemment, cette méthode est longue car il faut lire tout le fichier. Sur les fiches sélectionnées, il y a possibilité de sommer les rubriques numériques, mais attention ce n'est pas un calc. Le résultat n'est pas gardé, seulement affiché.

Particularité de ce forth là:

ACCENT et QWERTY permettent d'écrire l'un avec des minuscules accentuées, l'autre de revenir au standard
AT permet d'écrire à un endroit particulier de l'écran (voir LOCATE).
ONLY FORTH DEFINITIONS peut devenir FORTH DEFINITIONS.
Mais surtout la sortie sur imprimante: il faudra sûrement réécrire les mots des trois dernières lignes de l'écran 8; le PON signifie Printer-On et envoie en même temps l'affichage sur écran et l'impression.

Quand au LINIT il est encore plus particulier puisqu'il envoie uniquement sur imprimante (PRINT) des codes d'initialisation de mon imprimante BROTHER M-1009.

Voilà, c'est tout: oui je sais, je n'ai pas une écriture forth très forth puisque je ne met aucun commentaire ni sur la pile et que je suis assez radin en place.

Amicalement J-L SIRET

```
FILE: FBASE2.FTH / SCRN# 1
\  OUVERTURE
ONLY FORTH DEFINITIONS DECIMAL
VARIABLE RLOC   VARIABLE RLEN  VARIABLE RNO   VARIABLE REC/BLK
VARIABLE BLKCLE VARIABLE FNO   VARIABLE CODE VARIABLE NBLKCLE
VARIABLE -BL     VARIABLE PLOC  VARIABLE PMAX  VARIABLE TRAV
VARIABLE NBCLE  VARIABLE NTYPE VARIABLE FLOC  VARIABLE TAMPON
: PHIGH TAMPON @ @ ;
CREATE CLES 0 , 0 , 0 , 0 , 0 ,
: RECORD DUP RNO ! 1- 1024 RLEN @ / /MOD BLKCLE @ + NBLKCLE @ +
  BLOCK SWAP RLEN @ * + DUP RLOC ! ;
: IREC RLEN ! NBLKCLE ! BLKCLE ! ;
: FFIND 2* PLOC @ 1+ + @ DUP FLOC ! ;
: FLEN FLOC @ 4 + C@ ;
: FCLE FLOC @ 7 + C@ 0<> ;
: FTYPE FLOC @ 5 + C@ NTYPE ! ;
-->
```

```
FILE: FBASE2.FTH / SCRN# 2
\ INITIALISATION
: TBCLE 0 NBCLE ! 10 0 DO -1 CLES I + C! LOOP
  PMAX @ 0 DO I FFIND DROP FCLE IF NBCLE 10 <>
  IF I NBCLE @ CLES + C! 1 NBCLE +! THEN THEN LOOP ;
: NMCLE TRAV ! -1 BEGIN 1+ DUP CLES + C@ TRAV @ = UNTIL ;

: PINIT DUP PLOC ! C@ PMAX ! 32 -BL ! 0 FNO ! TBCLE HERE
  TAMPON ! 1024 NBLKCLE @ * ALLOT NBLKCLE @ 0 DO BLKCLE @ I +
  BLOCK TAMPON @ 1024 I * + 1024 CMOVE LOOP ;
: PBUILD CREATE DUP C, HERE PLOC ! DUP 2* ALLOT
  0 DO HERE PLOC @ I 2* + ! 8 0 DO BL WORD NUMBER DROP C, LOOP
  34 WORD C@ 1+ ALLOT LOOP DOES> ;

: FLOC1 RLOC @ FLOC @ 6 + C@ + ;
: POSIT FLOC @ DUP 2+ C@ SWAP 3 + C@ SWAP AT ;
-->
```

→

```
FILE: FBASE2.FTH / SCRN# 3
\ ENTREE FICHE (1)
: FDISPLAY DUP DUP C@ SWAP 1+ C@ SWAP AT 8 + COUNT TYPE
  POSIT FLEN 0 DO 45 EMIT LOOP ;
: PDISPLAY CLS PMAX @ 0 DO I FFIND FDISPLAY LOOP ;

: INPUT BLK @ >R >IN @ >R 0 BLK ! QUERY BL WORD NUMBER DROP
  R> >IN ! R> BLK ! ;
: WAIT CR CR ." Appuyez sur une touche pour continuer "
  KEY DROP ;
: ASK ." (O/N)? " KEY DUP ASCII O = SWAP ASCII o = OR ;

: D! 2DUP ! 2+ SWAP DROP ! ;
: D@ DUP 2+ @ SWAP @ ;
: CPLACE NBCLE @ RNO @ 1- * FNO @ NMCLE + 2* ;
: FCODE PAD C@ 32 - 100 * PAD 1+ C@ 32 - 10 * + PAD 2+ C@
  32 - + CCODE ! ;          -->
```

```
FILE: FBASE2.FTH / SCRN# 4
\ ENTREE FICHE (2)
: NUM1 PAD 1- SPAN @ OVER C! NUMBER? DROP ;
: FNUM FLOC1 NUM1 OVER CCODE ! ;
: FTEXT FLOC1 DUP FLEN BLANK PAD SWAP SPAN @ CMOVE FCODE ;
: ENTRE PAD DUP 128 BLANK FLEN EXPECT SPAN @ 0<> ;
: FGET DUP FNO ! FFIND DROP  FLEN 0<>
  IF POSIT FTYPE ENTRE IF NTYPE @ CASE 9 OF FTEXT ENDOF
                    1 OF FNUM DROP SWAP C! ENDOF
                    2 OF FNUM DROP SWAP !  ENDOF
                    3 OF FNUM DROP SWAP !  ENDOF
                    4 OF FNUM ROT D! ENDOF
                    5 OF FNUM ROT D! ENDOF
    ENDCASE FCLE IF TAMPON @ 2+ CPLACE + CCODE @ SWAP ! THEN
  THEN THEN ;
: PGET UPDATE PMAX @ 0 DO I FGET LOOP ;
-->
```

```
FILE: FBASE2.FTH / SCRN# 5
\  LECTURE FICHE
: PNEXT PHIGH 1+ DUP TAMPON @ ! RECORD DROP ;
: PADD PDISPLAY PGET ;
: .FR SWAP OVER DABS <# # # 46 HOLD #S SIGN #> TYPE SPACE ;

: AFFICH FTYPE FLOC1 FLEN NTYPE @ CASE
      9 OF TYPE ENDOF                1 OF SWAP C@ SWAP .R ENDOF
      2 OF SWAP @  SWAP .R ENDOF   3 OF DROP @ 0 .FR    ENDOF
      4 OF SWAP D@ ROT D.R ENDOF   5 OF DROP D@  .FR    ENDOF
  ENDCASE ;
: FPUT DUP FNO ! FFIND DROP POSIT AFFICH ;
: PPUT 10 0 AT ." Fiche " RNO @ 3 .R PMAX @ 0 DO I FPUT LOOP ;
: PSELECT CLS CR ." Entrez le numero d'enregistrement: " INPUT
  DUP PHIGH > IF CR ." Pas d'enregistrement " CR DROP 0 ELSE
  RECORD DROP 1 THEN ;
-->
```

```
FILE: FBASE2.FTH / SCRN# 6
\ SELECTION DE FICHES
VARIABLE CPTRI CREATE FICHTRI 256 ALLOT
: INITRI 256 0 DO I 1+ I FICHTRI + C! LOOP PHIGH CPTRI ! ;
: PREND FICHTRI + C@ ;
: MET PREND CPTRI @ FICHTRI + C! 1 CPTRI +! ;

: FRECH FNO @ NMCLE 2* TAMPON @ 2+ + TRAV ! CPTRI @ 0 CPTRI !
  0 ?DO I PREND 1- NBCLE @ 2* * TRAV @ + @ CCODE @ =
  IF I MET THEN LOOP ;

: FSAISI DUP FNO ! FFIND CR 8 + COUNT TYPE ."  :" ENTRE
  IF FTYPE NTYPE @ 9 = IF FCODE ELSE NUM1 DROP CCODE !
  THEN FRECH THEN ;
: PRECH CLS NBCLE @ 0= IF ." Cl{s de tri inexistantes" WAIT
  ELSE NBCLE @ 0 DO I CLES + C@ FSAISI CR LOOP THEN ;
-->
```

```
FILE: FBASE2.FTH / SCRN# 7
\ LISTE SELECTION PAR MASQUE ALPHA
: FICH? CPTRI @ 0= IF ." Aucune fiche s{lectionn{e" WAIT 0
        ELSE -1 THEN ;
: FLIST CLS FICH? IF CPTRI @ 0 DO PDISPLAY I PREND RECORD DROP
        PPUT WAIT CLS LOOP THEN ;

: COMPTEXT TRAV ON 1- BEGIN >R 2DUP R + R> ROT OVER + ROT
  DUP C@ 63 = IF 2DROP ELSE C@ SWAP C@ <> IF TRAV OFF THEN
  THEN 1- DUP -1 = TRAV @ 0= OR UNTIL DROP 2DROP TRAV @ ;

: FSAIS FFIND 6 + C@ FTYPE NTYPE @ 9 =
  IF POSIT PAD + FLEN EXPECT ELSE DROP THEN ;

: PSAIS PDISPLAY PAD RLEN @ 63 FILL PMAX @ 0 DO I FSAIS LOOP ;

-->
```

```
FILE: FBASE2.FTH / SCRN# 8
\ COMPARAISON MASQUE ET EDITION
: PCOMP CLS FICH? IF PSAIS CPTRI @ CPTRI OFF
        0 ?DO I PREND RECORD PAD RLEN @ COMPTEXT
        IF I MET THEN LOOP THEN ;

: PRUBR CLS PMAX @ 0 DO I FFIND 8 + COUNT TYPE SPACE ASK
  IF ." O" TRUE ELSE ." N" FALSE THEN PAD I + C! CR LOOP ;

: FEDIT PMAX @ 0 DO I PAD + C@ IF I FFIND DROP AFFICH
        SPACE THEN LOOP ;
: PEDIT FICH? IF CPTRI @ 0 DO I PREND RECORD DROP
        FEDIT CR LOOP THEN ;
ALSO PRINTER ALSO FORTH DEFINITIONS
: LINIT 1 82 27 15 60 27  6 0 DO (PRINT) LOOP ;
: LEDIT LINIT PON PEDIT POFF ; ONLY FORTH DEFINITIONS
-->
```

```
FILE: FBASE2.FTH / SCRN# 9
\ SOMME SUR RUBRIQUE NUMERIQUE
: INIPAD PAD PMAX @ 4 * ERASE ;
: FCOMPTE RECORD DROP PMAX @ 0 DO I FFIND DROP FTYPE FLOC1
  NTYPE @ CASE 9 OF DROP 0 0 ENDOF 1 OF C@ 0 ENDOF 2 OF @ 0
  ENDOF 3 OF @ 0 ENDOF 4 OF D@ ENDOF 5 OF D@ ENDOF ENDCASE
  PAD I 4 * + D@ D+ PAD I 4 * + D! LOOP ;

: RESULT PMAX @ 0 DO I FFIND FTYPE NTYPE @ 9 <> IF
  8 + COUNT TYPE SPACE PAD I 4 * + D@ NTYPE @ DUP 3 = SWAP 5 =
  OR IF .FR ELSE D. THEN CR ELSE DROP THEN LOOP ;

: PCOMPTE CLS FICH? IF INIPAD CPTRI @ 0 DO I PREND FCOMPTE LOOP
  RESULT WAIT THEN ;


-->
```

```
FILE: FBASE2.FTH / SCRN# 10
\ FIN
: PLIST CLS PHIGH 1+ 1 ?DO I 3 .R SPACE
  I RECORD DROP 2 0 DO I FFIND DROP FLOC1 FLEN -TRAILING TYPE
  SPACE LOOP CR I 23 MOD 0= IF CR WAIT CLS THEN LOOP WAIT ;

: PEND CR CR NBLKCLE @ 0 DO TAMPON @ 1024 I * + BLKCLE @ I +
  BLOCK 1024 CMOVE UPDATE LOOP ." Sauvegarde des fichiers" CR
  SAVE-BUFFERS ." Au-revoir" CR ;

-->
```

```
FILE: FBASE2.FTH / SCRN# 11
\ MENU DE TRI
: PMENU2 BEGIN CLS 10 2 AT ." Recherche/Tri/Affichage" CR
  10 3  AT ." Fiches s{lectionn{es : " CPTRI @ . CR
  6 6  AT ." 1 - Recherche par cles" CR
  6 8  AT ." 2 - Lister les fiches" CR
  6 10 AT ." 3 - Tri par masques " CR
  6 12 AT ." 4 - Edition sur {cran" CR
  6 14 AT ." 5 - Edition sur imprimante" CR
  6 16 AT ." 6 - Somme des rubriques num{riques" CR
  6 18 AT ." 9 - Retourner au menu pr{c{dent" CR
  6 20 AT ." Votre choix :" KEY DUP 57 <> IF
  CASE 49 OF PRECH 0 ENDOF 50 OF FLIST 0 ENDOF
        51 OF PCOMP 0 ENDOF 52 OF PRUBR CLS PEDIT WAIT 0 ENDOF
        53 OF PRUBR CLS LEDIT WAIT 0 ENDOF 54 OF PCOMPTE 0 ENDOF
  0 SWAP ENDCASE ELSE DROP -1 THEN UNTIL ;
-->
```

```
FILE: FBASE2.FTH / SCRN# 12
\ MENU FINAL
: MENU ACCENT BEGIN CLS 10 2 AT ." Maintenance de fichier "
  10 3 AT ." Nombre de fiches : " PHIGH . CR
  6  6 AT ." 1 - Ajouter"  CR  6 8 AT ." 2 - Lister" CR
  6 10 AT ." 3 - Afficher fiche" CR
  6 12 AT ." 4 - Changer  fiche " CR 6 14 AT ." 5 - Tri " CR
  6 16 AT ." 9 - Quitter" CR
  CR 5 20 AT ." Votre choix" KEY DUP 57 <> IF
  CASE 49 OF PNEXT PADD 0 ENDOF 50 OF PLIST 0 ENDOF
  51 OF PSELECT IF PDISPLAY PPUT THEN WAIT 0 ENDOF
  52 OF PSELECT IF PDISPLAY PPUT PGET THEN 0 ENDOF
  53 OF INITRI PMENU2 0 ENDOF
  0 SWAP ENDCASE ELSE DROP PEND -1 THEN UNTIL QWERTY ;

CR CR .( FBASE2 est compile ) CR
```

suite page 17

Si une erreur est décelée, un "dump" est effectué du début du tampon disque à la fin du mot erroné.
(SOURCE) est une version modifiée du standard. BLK est utilisé comme drapeau indiquant si le flot d'entrée provient du clavier ou du tampon disque.
?ERROR aussi est une version modifiée : au lieu de laisser les paramètres pour la commande WHERE, il "dumpe" le tampon.

Voici un exemple d'une session, où l'on suppose que 3 fichiers ont été créés : FILEA.BLK, FILEB.BLK, FILEC.BLK, et que les 2 dernières lignes de code dans FILEA.BLK sont :

```
CR  .( Chargement de FILEB.BLK )
LOAD FILEB.BLK
```

et que les 2 dernières lignes de FILEB.BLK sont

```
CR  .( Chargement de FILEC.BLK )
LOAD FILEC.BLK
```

Les définitions de UNSCREEN étant chargées, il suffit de faire

```
LOAD FILEA.BLK
```

pour obtenir l'interprétation (la compilation) des 3 fichiers. ■

Suite de la page 16

```
FILE: FBASEX2.FTH / SCRN# 1
( 0) \ FBASE1 EXEMPLE
( 1) 9 PBUILD ANNUAIRE
( 2) 01 02 01 09 31 9  00 1 NOM"
( 3) 03 02 03 09 22 9  31 0 PRENOM"
( 4) 05 02 06 02 38 9  53 0 ADRESSE"
( 5) 08 02 08 14 05 4  91 0 CODE POSTAL"
( 6) 10 02 11 02 38 9  96 1 COMMUNE"
( 7) 13 02 13 12 02 1 134 0 TELEPHONE"
( 8) 13 14 13 15 02 1 135 0 ."
( 9) 13 17 13 18 02 1 136 0 ."
(10) 13 20 13 21 02 1 137 0 ."
(11) \ La premiere fois initialisation
(12) \ 2 1 138 IREC 2 BLOCK 1024 ERASE UPDATE SAVE-BUFFERS
(13) \ A toutes les fois
(14) 2 1 138 IREC ANNUAIRE PINIT MENU
(15)
```

■

VOLKS-FORTH
FORTH 83-Standard
pour ATARI 260ST, 520ST, 520ST+

réalisé par FIG HAMBOURG (RFA)

Suite aux différents contacts pris entre l'ASSOCIATION JEDI et le FORTH INTEREST GROUP de HAMBOURG, Mr Bernd PENNEMANN nous a expédié un échantillon de la version F83 pour ATARI, nommé Volks-FORTH et, ce qui ne gâte rien, est 'public domain'. Il est accompagné d'une notice très complète et très dense écrite en allemand. Pour les volontaires traducteurs, FIG-HAMBOURG recherche des bénévoles pour traduire d'allemand en anglais ladite notice.

Pour notre part, l'ASSOCIATION JEDI diffusera les disquettes à la demande. Mais comme l'association propose déjà un manuel pour la version MSDOS et CP/M, nous ne ferons part que des différences existantes entre Volks-FORTH et les versions MSDOS-CP/M, nottament en matière de graphisme, DOS, etc... De plus, FIG-HAMBOURG accepte que nous reprenions dans notre magazine des programmes réalisés pour V-FORTH (c'est plus court que Volks-FORTH) et facilement adaptables au F83 MSDOS-CP/M.

PROLOGUE

Tout comme la version F83 de Laxen et Perry, diffusée pour les système MSDOS et CP/M, V-FORTH est du domaine public et s'inspire largement de la syntaxe de F83. Il a au départ été conçu pour les systèmes COMMODORE 64 (C64), machine très répandue en Allemagne Fédérale. A l'apparition des premiers ATARI de la série ST, V-FORTH y a été adapté. Le système de traçage de l'exécution d'une routine a été amélioré, les outils spécifiques ST et la bibliothèque GEM créés. Dans la version actuelle 3.8, l'éditeur GEM est disponible, le code est relogeable, permettant ainsi un développement aisé des applications.

POURQUOI PROGRAMMER AVEC V-FORTH83 ?

V-FORTH est un outil incomparablement pratique et compact. La librairie des routines exécutables, le compilateur, l'éditeur et le débuggeur sont disponibles en permanence, rendant superflu le cycle fastidieux ECLG ("Edit, Compile, Link and Go"). Le code peut être réalisé, compilé et testé module par module. Le débuggeur intégré est idéal pour tester l'exécution d'une définition; il n'y a pas besoin de faire appel à un dump hexa ou à un listing d'assemblage démesuré et n'ayant que peu de rapport avec le texte source.

L'autre aspect est la gestion multi-tâche. Ainsi, un programme peut être partagé en mots ou modules indépendants et dont l'exécution est autonome. Ceci n'est pas concevable dans la majorité des langages.

Enfin, V-FORTH dispose d'une foule de détails dont ne disposent pas les autres systèmes FORTH.

V-FORTH est disponible en trois disquettes et dispose d'un traceur, décompilateur, système multi-tâche, éditeur, assembleur et désassembleur, bibliothèque GEM, demos graphiques, interface impression...

Encore une chose: vous pouvez céder V-FORTH à vos amis sans rendre de compte. Mais si quelqu'un le cède contre argent avec bénéfice, nous lui en tiendrions rigueur jusqu'au bout du monde et à la fin de ses jours!

Le support technique et logiciel de V-FORTH est assuré par FIG HAMBOURG et son magazine "VIERTE DIMENSION - Forth Magazin", son réseau FORTH avec boîtes aux lettres:

FORTH GESELLSCHAFT e.V.
Friedensalle 92
D - 2000 HAMBURG 50 (RFA/BRD)
tel (19/49) 040 - 390 42 04 ligne directe mardi de 18h à 20h. NE PAS COMPOSER LE 0 de 040 depuis la France (liaison V21-V23 7bits - 300, 1200 bds).

Envoyez-nous vos remarques, programmes terminés ou inachevés, idées concernant V-FORTH, articles publiables, bref: envoyez tout ce qui concerne V-FORTH. Et naturellement: compléments, corrections et erreurs éventuelles concernant le manuel V-FORTH. (Ndlr: on n'y manquera pas).

Si V-FORTH vous plait, encouragez l'auteur, montant 20 DM,-, car la production de V-FORTH comme celle du manuel n'est pas gratuite, cette somme permettant de couvrir les frais.

Pour l'auteur de VolksFORTH:

Bernd PENNEMANN
Steilshooper Str. 46
D - 2000 HAMBURG 60 (RFA/BRD)

Le manuel a été réalisé par Bernd PENNEMANN, Klaus SCHLEISIEK, Georg REHFELD, Dietrich WEINECK, membres de FIG-HAMBOURG.

1ere PARTIE ON DEMARRE

Pour bien comprendre l'utilisation de VolksFORTH83, il faut étudier attentivement le présent manuel. Afin de progresser plus rapidement, nous vous décrirons l'essentiel dans ce chapitre:

- comment manipuler les disquettes
- comment démarrer le système
- comment créer une application définitive
- comment définir son propre environnement de travail.

MANIPULATION DES DISQUETTES

Avec votre manuel, vous avez reçu trois disquettes. Pensez toujours à en faire des copies de sécurité. Le risque de perdre des fichiers est grand, évitez de laissez votre FORTH entre toutes les mains - verrouillez les disquettes en écriture!! Sur une des disquettes se trouve le programme 4TH.PRG, et FORTHKER.PRG ainsi qu'un texte de documentation et le programme de démonstration 'Super copy'. Cette disquette sera nommée la disquette système.

4TH.PRG est le système de travail normal, FORTHKER.PRG une version minimale ne contenant que le noyau du langage. Avec celle-ci, vous pouvez définir une version FORTH avec éditeur personnalisé ou d'autres fonctions graphiques puis avec SAVESYSTEM la sauvegarder comme version définitive. Dans le même ordre d'idée, vous pouvez créer une application définitive où 'le tronc FORTH' n'apparait plus (un exemple en est SUPER COPY sur la disquette système). 4TH.PRG est un système de travail complet disposant de l'interface de gestion de fichiers, l'éditeur, l'assembleur, les outils, etc...

Les deux autres disquettes contiennent le texte source du système. Taper FILES (analogue à WORDS) pour la liste des fichiers. Dans le fichier FORTH 83.SCR figure le texte source du noyau du langage FORTH. Entendu que ce texte est destiné à être traité par un méta-compilateur, il est le reflet exact de FORTHKER.PRG. Vous pouvez voir le compilateur, et si vous le désirez, voir le fonctionnement de VolksFORTH83. Dans le fichier STARTUP.SCR se trouve un bloc de chargement, chargé de compiler les différentes parties appartenant à 4THPRG. A partir de ce bloc de chargement, 4TH.PRG est construit en complétant FORTHKER.PRG.

PREMIERS PAS

Mettez la disquette système dans le lecteur B, le texte source du fichier 2 dans le lecteur A (si vous ne disposez que d'un seul lecteur, il faudra procéder à des substitutions de disquettes). Lancez 4TH.PRG en cliquant à l'aide de la souris. VolksFORTH83 s'annonce accompagné de son numéro de version.

Tapez maintenant:

```
use tutorial.scr      (et return)
1 1                   (et return)
```

A la question 'Enter your ID :' répondez par appui sur RETURN. Vous vous trouvez maintenant dans l'éditeur et vous pouvez maintenant consulter l'écran courant et les suivants et apprendre les rudiments des manipulations de l'éditeur. L'éditeur fonctionne intégralement sous environnement GEM, ce qui ne vous changera guère de vos habitudes. Cependant, vous trouverez des explications complémentaires à la suite. Testez un peu les commandes, puis quittez l'éditeur en tapant sur (ESC).

Vous pouvez maintenant voir la démo-graphique. Elle se trouve sur la même disquette. Mais pour la faire fonctionner, il faut utiliser l'assembleur de la disquette 1. Pour celà sortez la disquette système du lecteur B et introduisez à la place la disquette des fichiers 1. Tapez:

```
include demo.scr      (et return)
```

Le lecteur disque tourne, vous voyez quels sont les blocs compilés puis quelques exemples graphiques. Après chaque image, passez à la séquence suivante par simple appui sur une touche ou tout interrompre par appui sur (ESC).

ET MAINTENANT LA SUITE...

Nous souhaitons maintenant compiler quelques mots et pour ce faire créer un nouveau fichier source. Pour ce faire, introduisez une nouvelle disquette dans le lecteur A. tapez ensuite:

```
makefile first.scr  2 more     (et return)
```

ce qui crée le fichier nommé FIRST.SCR d'une taille de deux blocs (2048 octets), contenant les écrans numérotés 0 et 1. Pour introduire des définitions dans le bloc écran 1, appeler à nouveau l'éditeur en tapant

```
1 1     (et return)
```

Dans la partie haute de l'écran on introduira une courte description du contenu de l'écran. Ainsi, avec le mot INDEX pourra-t-on se faire une idée du contenu d'un fichier. Ecrivez donc en commençant en début de ligne

```
' Mon premier programme FORTH
```

Le signe ' (backschlash) sert, en empèchant la compilation du contenu de la ligne situé à la suite de ce signe, à signaler un commentaire. Vous avez certainement remarqué que la date et les initiales des auteurs de programmes figuraient à droite de cette même ligne dans les autres fichiers. Cette identification est inscrite automatiquement par l'éditeur lors du premier appel de ID. Vous pouvez installer votre propre identification en tapant simultanément sur les touches CTRL-G ou en sélectionnant sous le menu GET-ID.

Laissons la seconde ligne libre, dans la troisième ligne définissons:

```
: test  ." Ceci est mon premier 'programme'." ;
```

Quittons l'éditeur en tapant CTRL-S. Le bloc ainsi modifié est écrit sur le disque. Appelons maintenant le compilateur par:

```
1 load          (et return)
```

A une vitesse équivalente à 'C' ou 'Pascal' notre mini programme est compilé et est prèt à être exécuté. Tapons, pour voir, le mot WORDS, afin de s'assurer que votre nouveau mot TEST figure bien en tête du dictionnaire (taper sur (ESC) pour interrompre l'exécution de WORDS, ou sur une autre touche pour le stopper temporairement). Pour tester l'exécution de notre premier mot, tapons:

```
test    (et return)
```

et là, voyez-vous, il se passe quelque chose. C'est joli, mais nettement insuffisant pour les pros que vous êtes, aussi supprimons le mot TEST précédemment compilé en tapant:

```
forget test
```

rappelons l'éditeur en tapant V . Devant la chaîne nous allons mettre CR. Le mot TEST ressemble maintenant à:

```
: test cr ." Ceci est mon premier 'programme'." ;
```

puis le recompilons. Ceci vous montre combien il est aisé d'écrire en FORTH, tester et faire exécuter des parties de programmes.

DEVELOPPER UNE APPLICATION

Nous voulons maintenant étoffer notre 'programme'. Pour ce faire, nous allons le grossir un peu (appeler l'éditeur en tapant V). Placez-vous à la suite de la définition de TEST et rentrez:

```
: runalone test key drop bye ;
```

RUNALONE exécute d'abord TEST, attend l'appui sur une touche et renvoie au moniteur. Compilez cette nouvelle définition, mais n'exécutez pas RUNALONE, sinon vous quittez FORTH.

VolksFORTH83 est prévu pour résoudre deux situations. Dans les mots COLD et RESTART se trouvent deux vecteurs nommés 'COLD et 'RESTART qui en temps ordinaire ne font rien, mais peuvent être intialisés ultérieurement. Nous manipulerons ici 'COLD. Pour ce faire tapons:

```
' runalone  is 'cold    (et return.. on ne le
                         répètera plus)
```

et sauvegardez le tout sur disquette en tapant

```
savesystem myprog.prg
```

Rangez ensuite MYPROG.PRG en cliquant. Vous venez de créer votre première application!

Cependant, le FORTH primitif pourtant très compact nécessite un espace mémoire d'au moins 40k, ceci pour afficher une ridicule petite chaîne de caractères ?? Quelque chose cloche. Naturellement, nous avons sauvegardé en même temps le système FORTH, la gestion disque, l'assembleur, la moitié de la bibliothèque GEM, l'éditeur, etc... qui ne sont certainement pas utiles pour notre programme.

Pour régler ce problème, nous disposons du fichier FORTHKER.PRG. Ce programme ne contient que le noyau du système et l'interface de gestion de fichier. Chargez FORTHER.PRG et compilez votre application en tapant:

```
include first.scr
```

Puis comme précédemment, relions RUNALONE à 'COLD et sauvegardons le système sur disque. Vous disposez maintenant d'une version compacte. Naturellement, cette version peut encore être simplifiée, mais pour celà il faut disposer d'un méta-compilateur lequel ne compilera que l'essentiel des parties du système utilisées par notre programme.

Selon le même principe on peut aussi compiler des programmes plus ambitieux et les sauvegarder commes programmes personnalisés.

POUR LES CHEVRONES DU FORTH

On peut, comme dans le précédent exemple, sans faire appel au méta-compilateur, récupérer 4k d'espace mémoire. Charger d'abord FORTHKER.PRG puis entrez:

```
' blk@ Is makeview
' noop Is custom-remove
' STr/w Is r/w
  direct
```

Ainsi les mots vectorisés gérant l'interface de gestion de fichers sont déconnectés et remplacés par l'accès direct.

Le premier mot de l'inteface de gestion de fichiers situé après SAVESYSTEM est DOS. Mais ce mot ne peut être oublié par FORGET, car situé dans la partie dictionnaire protégée. Faites vos essais avec prudence; vous recevrez le message 'DOS is protected'. Pour régler cette situation, nous disposons du mot (FORGET

```
' Dos >name 4- (forget   save
```

Ainsi vous disposez d'un système sans 'superflu'. Sauvegardez-le sur disquette en tapant

```
savesystem minimal.prg
```

Naturellement, nous ne pouvons plus charger notre fichier d'essai avec INCLUDE, car il n'y a plus d'interface de gestion de disque. Pour ce faire, nous rechercherons les blocs correspondants en accès direct. Chargeons à nouveau 4TH.PRG, déconnectons la gestion des fichiers en tapant

```
direct
```

puis voyons le contenu du disque par

```
1 500 index
```

(500 c'est peut être un peu beaucoup, mais INDEX s'arrète automatiquement à la lecture du dernier bloc disquette ou fichier.)

Notre écran d'essai doit certainement figurer dans un des blocs, mais nous prenons connaissance également du contenu de la disquette. Prenez note du numéro n du bloc contenant l'écran test et compilez son contenu sans faire appel à un nom de fichier en tapant

```
n load
```

INSTALLATION D'UN SYSTEME PERSONNALISE

Le programme 4TH.PRG est conçu en tant que version de travail. Il contient toutes les parties essentielles telles que l'éditeur, l'inteface imprimante, les outils, le graphisme GEM, le décompilateur, le débuggeur etc... Si vous ne désirez pas disposer de toutes ces fonctions, vous pouvez à tout moment créer votre version adaptée. Les clés de ce mécanisme fifguernt dans le fichier STARTUP.SCR sur la disquette 1. Oblitérez le chargement des nctions non souhaitées en les faisant précéder la ligne correspondante d'une barre de fraction inverse ou en effaçant le contenu de toute la ligne. De même vous pouvez inclure des fchiers personnels.

Lorsque le bloc de chargement correspond à vos souhaits,sauvegardez-le en tapant CTRL-S et quittez le système par BYE. Chargez ensuite FORTHKER.PRG, puis tapez:

```
include startup.scr
```

Ce qui compile votre système complet et personnalisé. Introduisez la disquette système et tapez

```
savesystem 4TH.PRG
```

et après avoir pensé à sauvegarder votre ancienne version de 4TH.PRG (toujours des copies de sécurité!!! - sicherheitskopie!!!). Au prochain chargement de 4TH.PRG, vous disposerez de 'votre' système. Naturellement, vous pouvez aussi sauvegarder votre système sous un autre nom à l'aide de SAVESYSTEM.

De même lors de la sauvegarde de votre système personnalisé, vous pouvez modifier la base numérique par

```
hex
```

ce qui peut être utile dans certains cas. Cette modification de base numérique est sauvegardée, puis conservée lors du rechargement du système (ce qui permet d'estimer, pour exemple, si l'espace mémoire 978584 réside dans la mémoire vidéo ou non?) avec SAVESYSTEM, comme décrit précédemment.

A signaler cependant que le chargement des blocs écrans d'un fichier risque d'être un peu perturbé si le chargement s'effectue dans une base numérique différente de la base décimale. Comme les nombres n'exploitent pas de préfixes $ pour hexadécimal, & pour décimal et % pour binaire, il sera plus judicieux de contrôler la base numérique dans chaque partie de programme concernée. Vous trouverez de nombreux exemples dans les écrans source.

Si vous ne disposez que d'un seul lecteur disquette, verrouillez la recherche multi-lecteur (normalement installée pour rechercher sur lecteur A, puis sur lecteur B) en activant la séquence

```
path ; a:
```

et sauvegardez cette adaptation par SAVESYSTEME.

SAUVEGARDE SYSTEME SUR DISQUETTE DOUBLE FACE

Si vous disposez d'un lecteur de disquette double face, il est possible d'y sauvegarder le système. Le problème est, qu'en cas d'accès à un fichier, celui-ci ne soit partagé sur deux faces. Il est conseillé de mettre toutes les parties du système sur une seule face. Pour ce faire, précisez par DIR et PATH l'ordre de recherche sur disque et ses limites.

On peut aussi travailler sur RAM-disk, mais ce n'est pas conseillé. FORTH est très près de la machine et en cas de plantage, vous perdriez toutes les informations. Vous pouvez travailler en RAM-disk en chargeant les commandes contenues dans le fichier RAMDISK.SCR. Ainsi vous pourrez stocker de nombreux blocks en RAM (en plus de FORTH), ce qui diminue les temps d'accès disque. En cas d'édition, mettez à jour le contenu du disque physique en transférant fréquemment le contenu du RAM-disk sur une disquette, ceci après un CTRL-S. Vous éviterez ainsi les désagréments d'une disparition de vos fichiers toujours possible.

IMPRESSION D'UN FICHIER SOURCE

Il peut être très utile d'imprimer le texte d'un fichier source, pour prendre connaissance nottament de la manière dont on débute avec FORTH. En dehors des fichiers source illustrés par de nombreux commentaires, vous trouverez des renseignements très intéressants dans README.DOC.

Chargez d'abord l'interface imprimante en tapant:

```
include printer.scr
```

Pour imprimer un fichier avec ses écrans commentaires associés, on tape:

use <u>nom-de-fichier</u> listing

Pour imprimer un fichier sans ses écrans commentaires associés, on tape:

use <u>nom-de-fichier</u> printall

... à suivre...

